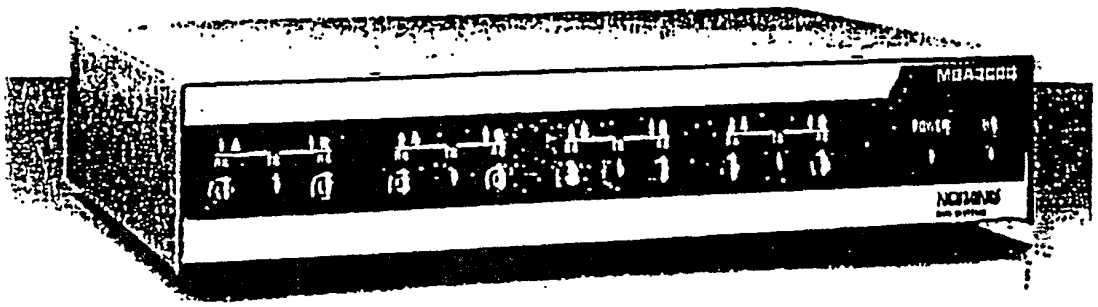
APPENDIX A

Brochure entitled "MBA3000 Multiple Base Adapter" (Two Sides)

The NORAND™ MBA3000 Multiple Base Adapter Dramatically Extends the Radio Frequency Coverage Area

MBA3000 Multiple Base Adapter



SYSTEM FEATURES

- Increases UHF coverage range by up to 100%
- Dual independent receiving improves contention response time during heavy RF transmission activity
- Supports multiple remote warehouses over standard telephone lines
- Dynamic time-division multiplexing, simulcast transmissions, and dual independent receiving provides the best possible RF coverage for the most difficult RF environments
- Allows up to 8 base stations to be operated when used in conjunction with the NORAND® RC2250, RC3240, and the RC3250 Network Controllers

The new MBA 3000 Multiple Base Adapter from Norand Corporation increases the RC2250, RC3240, or RC3250 Network Controller UHF base support from 2 bases to up to 8 bases. This increased support increases UHF coverage range by up to 100% and is ideal for multiple remote warehouse facilities.

Dynamic time division multiplexing insures adequate response time no matter where the RF terminal is located. The Multiple Base Adapter implements simulcast transmissions to allow for increased range without loss of user response time. Dual independent receiving improves contention response time during heavy RF transmission activity.

The MBA3000's operation is controlled by the RC2250, RC3240, or RC3250 Network Controller. It is a free standing unit that can be stacked or mounted horizontally.

The combination of dynamic time-division multiplexing, simulcast transmissions, and dual independent receiving provides the best possible RF coverage for the most difficult RF environments.

NORAND®
DATA SYSTEMS

APPENDIX B

Brochure entitled "RB2212 Base
Radio Transceiver" (Two Sides)

NORAND™
DATA SYSTEMS

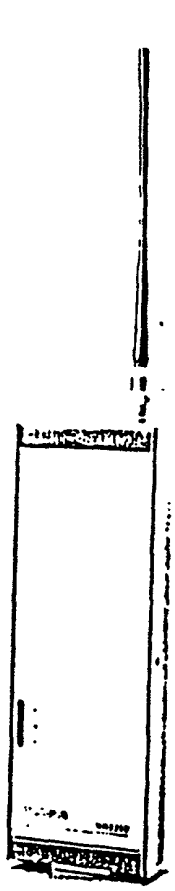
RB2212 Base Radio Transceiver

The NORAND™ RB2212 Base Radio Transceiver is the Wireless Interface to Your Host Computer

Portable radio data terminals constitute the latest technological advance in the utilization of computer power. Radio terminals add a new dimension to computers by extending their power to control transactions where they occur. But utilization of radio terminals can only be successful if communications with the host computer are reliable. The RB2212 Base Radio Transceiver from Norand is designed to provide a radio communication link as reliable as a direct cable connection to your host computer.

To accomplish this, we designed the base transceiver with a high-performance, digital data radio instead of modifying a radio designed for voice communications. This ensures high-speed, reliable data communications between the host computer and the radio data terminals. And the RB2212 is designed to operate in rugged environments, allowing an extended range of radio communications beyond the point where modified voice radios will fail.

Add up the features and you'll understand why the RB2212 and all of the radio data products from Norand set the industry standard for performance and reliability.



FEATURES:

- High-performance radio designed for data communications
- Operational status indicators
- High-speed radio communications
- System compatibility with all NORAND® RF hand-held terminals

RB2212 Base Radio Transceiver SPECIFICATIONS

• The Radio Data Network

The NORAND® RB2212 Base Radio Transceiver is a component of a complete radio data network. The network consists of a communications multiplexer, the RB2212 Base Radio Transceiver, and up to 16 radio data terminals. The radio data network extends your computing power by providing interactive communications between the host computer and the remote areas of your facility.

The portable element of the radio data network is the radio data terminal. Terminals are carried throughout your facility to put the power of your host computer where the action is.

The terminals communicate with the host computer via the RB2212 Base Radio Transceiver and the communications multiplexer. The RB2212 is the radio link to the terminals, utilizing a high-performance digital data radio to ensure reliable communications.

The RB2212 is controlled by the communications multiplexer, which manages the network communications and direct data transfer between the terminals and the host computer. The communications multiplexer communicates with the host computer via an RS232 interface cable.

Product Features:

Indicators: Light Emitting Diode (LED) indicators reflect transceiver status (TXD [transceiving], RXD [receiving])

Case: Two toned (industrial dark gray and industrial light gray) aluminum wall mounted case provides excellent electrical shielding and ground plane for the radio

Radio Characteristics:

Radio Transceiver: Compact, high performance digital data radio

Radiated Power: 1 watts nominal

Frequency Range: 1 to 120 MHz

Physical Dimensions:

Size: 13" x 10" x 5" (LWH) (33 cm x 25 cm x 13 cm)

Weight: 11 pounds (5 kg)

Environmental Characteristics:

Temperature:

Operating: 14° to 122°F (-10° to 50°C)

Storage: -22° to 158°F (-30° to 70°C)

Humidity: 10 to [redacted] noncon[redacted]ing

Altitude: To 10,000 feet (3,048 meters) above sea level

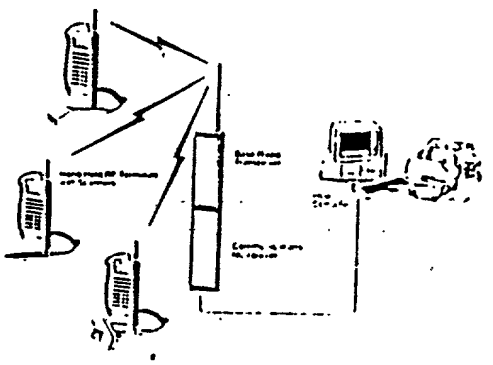
NORAND™
DATA SYSTEMS

Norand Corporation
550 Second Street S.E.
Cedar Rapids, Iowa 52401
Phone: 319/369-3156
1-800-553-5971 toll free (ext. 3156)

Norand Data Systems, Ltd.
751 Denison Street
Unit #1
Markham, Ontario
Canada L3R 3W9
Phone: 416-477-1818

Norand (U.K.) Ltd.
5 Bennet Court
Bennet Road
Reading, Berkshire RG2 0QX
England
Phone: (44[redacted]-861221

Radio Data Network Flow



APPENDIX C

Brochure entitled "RB3000 Base
Radio Transceiver" (Two Sides)

NORAND
DATA SYSTEMS

RB3000 Base Radio Transceiver

The NORAND™ RB3000 Base Radio Transceiver is the Right Solution for Industrial/Warehouse Applications

As the use of radio data terminals increases, ways to improve throughput while providing a reliable radio link to the host are being sought. The RB3000 Base Radio Transceiver from Norand achieves these goals.

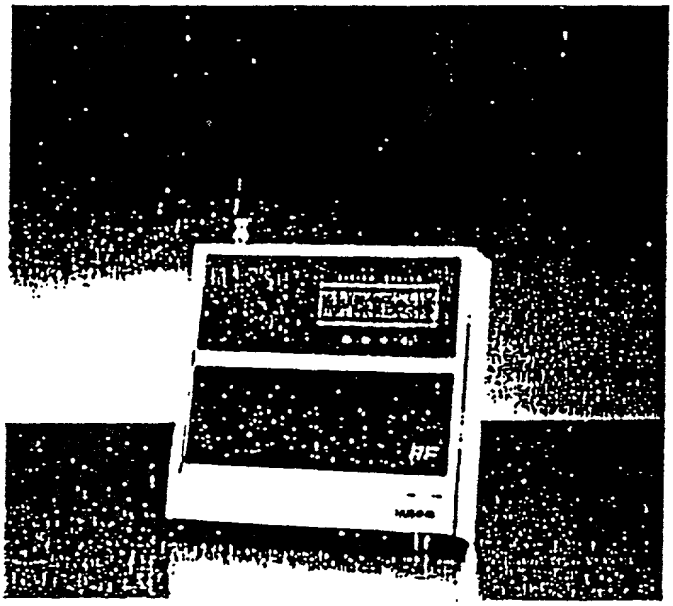
Developed for industrial and warehouse applications, where voice radio systems are often already in use, the RB3000 was designed to reject unwanted signals from other systems and to be less likely to interfere with neighboring radio networks.

The RB3000 utilizes a third-generation, high-performance digital transceiver enhanced for 9600 bit-per-second (BPS) operation. The RB3000 can provide better throughput than a 4800 BPS base radio when combined with our patented Real-Time Control™ radio link communication protocol and baud rate switching. It provides improved throughput by compensating for the range and error rate problems associated with radio communications at 9600 BPS. And we've maintained compatibility with all NORAND RF hand-held terminals.

Consider the importance of reliable, fast communications to your operation and you'll see why the RB3000 is the best solution.

• The Radio Data Network

The NORAND RB3000 Base Radio Transceiver is a component of a complete radio data network. The network consists of a communications multiplexer (or controller), the RB3000 Base Radio Transceiver and up to 127 terminals. The maximum number of terminals depends on the type of multiplexer or controller selected and whether the host interface is asynchronous or SNA/SDLC.



The portable element of the system is the radio data terminal. Terminals are carried throughout your facility to put the power of your host computer where the work needs to be done. Norand offers a variety of terminals designed for different operating conditions.

The terminals communicate with the host computer via the RB3000 Base Radio Transceiver and communications multiplexer (or controller). The RB3000 utilizes a high-performance digital data radio to ensure reliable communications.

The RB3000 is controlled by the communications multiplexer (or controller) which manages the radio network communications and data transfer between the terminals and the host computer. The communications link between the multiplexer (or controller) and the RB3000 can be via an RS232 or RS422 interface cable.

FEATURES:

- High-performance third-generation radio designed for data communications
- Patented Real-Time Control™ protocol and baud rate switching for reliable 4800 BPS or 9600 BPS communications
- System compatibility with all NORAND® RF hand-held terminals
- Operational status indicators

RB3000 Base Radio Transceiver
SPECIFICATIONS

Product Features:

Indicators: Light Emitting Diode (LED) indicators reflect transceiver status (ON [power on], TX [transmitting], RX [receiving], HS [9600 BPS operation])

Radio Link Transmit Speed (Bits-Per-Second): 4800 (with communications multiplexer), 9600 (with network controller), 4800/9600 Baud Rate Switching* (with network controller)

Base Radio Serial Interface:
RB3000 - RS232
RB3001 - RS422

Case: Molded polymer wall-mounted case

Radio Characteristics:

Radio Transceiver: High-performance, third generation digital data radio enhanced for 9600 BPS operation

Radiated Power: 2 watts nominal

Frequency Range: 150 to 170 MHz

Physical Dimensions:

Size: 11.5" x 2.5" x 10.2" (LWH) (29.5cm x 6.4cm x 26.2cm)

Weight: 3 pounds (1.36kg)

Environmental Characteristics:

Temperature:

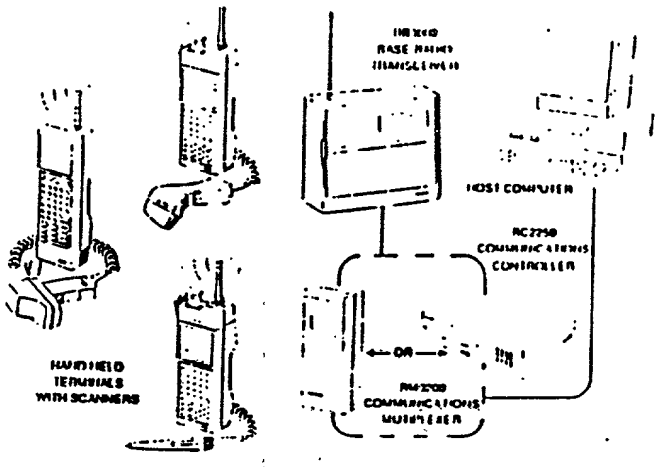
Operating: -4° to 122°F (-20° to 50°C)

Storage: -22° to 140°F (-30° to 60°C)

Humidity: 10 to 90% noncondensing

Altitude: To 10,000 feet (3,048 meters) above sea level

Radio Network Data Flow



* Patented in the United States. Patents applied for in other countries.

NORAND DATA SYSTEMS

Norand Corporation
550 Second Street S.E.
Cedar Rapids, Iowa 52401
Phone: 319-369-3156
1-800-553-5971 toll free (ext. 3156)

Norand Data Systems, Ltd.
951 Denison Street
Unit #4
Markham, Ontario
Canada L3R 3W9
Phone: 416-477-1818

Norand (U.K.) Ltd.
5 Bennet Court
Bennet Road
Reading, Berkshire RG2 0QX
England
Phone: (44) 734-861221

APPENDIX D1

Brochure entitled "RT2210XL Radio
Data Terminal" (Six Sides)

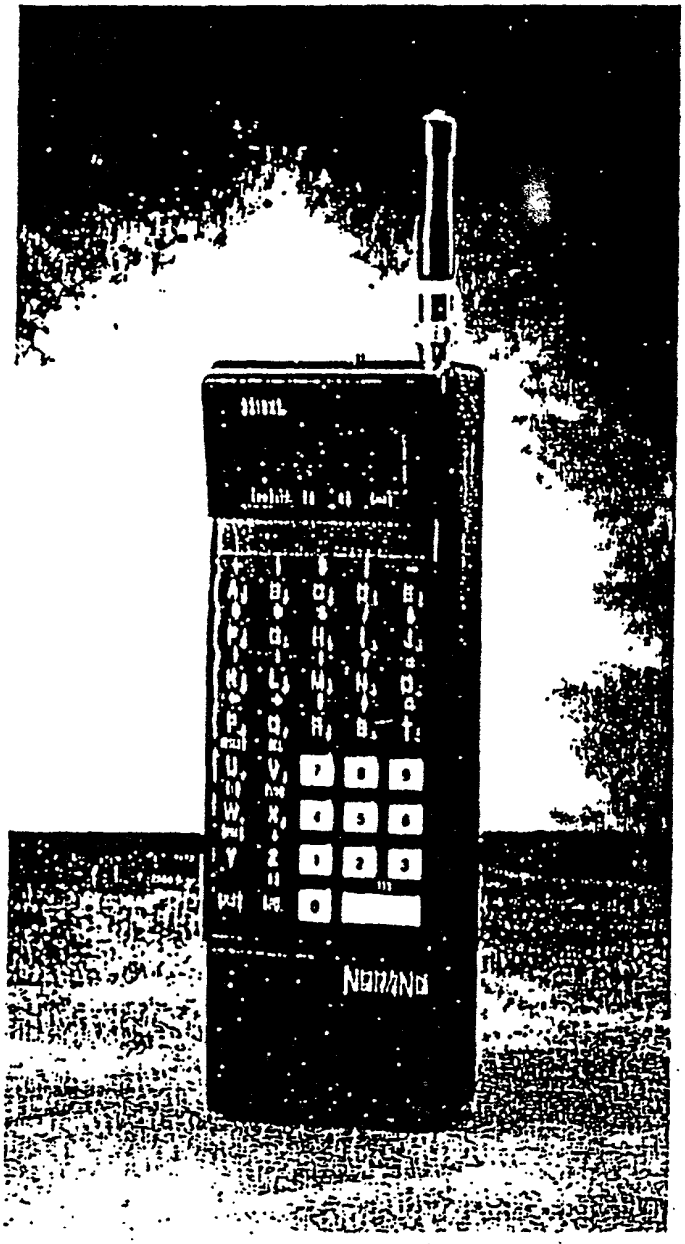
NORAND
DATA SYSTEMS

RT2210XL
Radio Data Terminal

The NORAND® RT2210XL Radio Data Terminal Puts the Power of Your Host Computer Where You Need it Most

The RT2210XL Radio Data Terminal from Norand is a wireless, interactive terminal which provides a new freedom in data collection and communication. The newly designed, ruggedized terminal extends the benefits of computerization by putting the power of your computer wherever you need it the most.

The terminal provides the link between your host computer and remote areas of your facility. The combination of data communications and two-way FM radio technology allows you to control transactions as they occur. The addition of high-speed bar code scanning gives you a real-time data collection system that delivers unsurpassed, bottom line results.



FEATURES:

- Lightweight, wireless portability with two-way, interactive data communication capabilities
- Compact, ruggedized design
- High performance radio providing optimum coverage for data communications
- High resolution display (64-character) with backlighting
- Bar code scanning support

The RT2210XL Has the Features You're Looking For

• **Wireless Portability**

Traditionally, the only access to a computer was from a hard wired terminal station. As computers take on a greater portion of the inventory, accounting, and purchasing burden, the need to enter information into the host computer from remote areas of a facility becomes even more apparent. The RT2210XL gives you the portability that you need, while maintaining the data integrity of the hard-wired computer terminal.

• **High-Performance Data Radio**

The NORAND® RT2210XL uses a high-performance, second generation data communication radio, different from the radio devised for voice communications. The Norand radio is specially designed to optimize data transmission/reception. It performs where voice radios fail.

• **Two-Way Interactive Data Communications**

The RT2210XL Radio Data Terminal permits two-way data communications between the terminal and the host computer. Whether it's providing lightning fast price verification in a retail store or inventory updates in an industrial setting, interactive data communications deliver results.

• **Compact, Ruggedized Design**

The digital components, radio, a 39 key alphanumeric keyboard, and battery pack of the RT2210XL are housed in a compact, ruggedized package which fits in one hand. This lightweight unit weighs only 2 pounds (907g), making it ideal for operating hours without fatigue.

• **Liquid Crystal Display (64-Character)**

The user of the hand held terminal can view up to 4 lines by 16 characters each on the 1 inch x 2.25 inch (2.5cm x 5.72cm) display. Features of the display include large, easy to read characters and four status indicators, allowing you to determine the terminal's state of operation at a glance. The display also incorporates back-lighting for low light environments or nighttime operation.

Real-time inventory control in industrial environments can be accomplished quickly and easily using the NORAND® RT2210XL Radio Data Network.

Ensuring pricing accuracy is one of the many advantages of using the RT2210XL Radio Data Network in the retail industry.

• **Bar Code Scanning Support**

The hand held RT2210XL supports all major bar code symbologies and is designed to interface with a wide variety of bar code scanning devices. NORAND 20/20 CCD Bar Code Scanners, light pens, LS8110 Laser Scanners, as well as various third-party scanners are all supported by the standard RT2210XL.

An optional direct-laser scan feature is also available for 5-volt helium-neon laser and laser-diode scanners like our LS7000, LS8100, and LS8500 Laser Scanners (as well as compatible third-party scanners). This option eliminates the need for an external communication interface or power supply.

• **Available Options**

To meet individualized needs, several options are available to complement the RT2210XL's standard features. These options include:

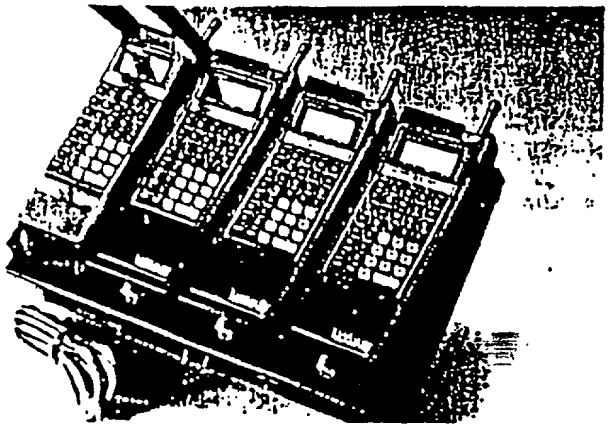
- direct laser scanning capability.
- non-volatile memory.
- numeric keyboards (20-key).

Summary of NORAND® RT2210XL Radio Data Network System Features

FEATURE	DESCRIPTION	ADVANTAGES
Single-unit integrated design	The two-way FM radio and the computer are built into a single unit, lightweight design. The ruggedized RT2210XL requires no external communication interfaces or power supplies.	The RT2210XL goes easily to where data collection is required. Its small size and lightweight construction are perfect for extended, hand held operation without fatigue. In addition, there are no tethering cables to get tangled.
Two-way interactive data communications	RT2210XL Radio Data Terminals provide instantaneous host computer access for data collection, update, and inquiry applications.	On line computer power is available "where the action is." Personnel on the move can enter data at the source while having access to the host computer applications. Since the accuracy of data entries can be immediately verified by the host computer, errors are eliminated at the transaction level.
Nicad battery power supply	Rechargeable, nickel cadmium battery packs provide reliable operating power for the RT2210XL.	No external power source is required. One battery pack can provide up to 10 hours of uninterrupted use. The battery packs are interchangeable and easy to replace.
High-performance digital data radio	A uniquely designed high performance digital data radio is the backbone of the RF data communications network.	Utilizing a radio designed specifically for two-way, digital data communications assures more reliability in areas where marginal coverage exists with fast, reliable message reception and high performance make the RT2210XL RF design the industry standard in excellence and value.
Bar code scanning support	All major bar code symbologies are supported in the RT2210XL. The RT2210XL also supports a variety of bar code scanning devices including CCD scanners, light pens, direct laser and diode laser scanners.	Use of the latest in bar code identification technology provides the fastest, most accurate method of data entry. Combining scanning technology with RF data communications provides the ultimate control over operations.
Built-in audio annunciator	A built in audio alert system may be used to inform you of operational conditions.	Audible feedback alerts the operator to take appropriate actions or signals the operation of a completed operation. Various uses of this feature include inbound message reception, full buffer alert, improper data entry, and properly/improperly decoded bar code.
Buffer data storage	The RT2210XL has 8K bytes of data storage where multiple prompts and data entries are retained until they can be accurately transmitted to the host computer.	Operators may review previous entries prior to sending the data to the host computer by scrolling through the data buffer.
Operational status indicators	Indicators at the bottom of the Liquid Crystal Display (LCD) inform the user of how the unit is configured, keyboard shift mode, radio transmission, and radio reception.	Provides the user with visual feedback of terminal operating conditions and indicates corrective action.
LCD with backlighting	The LCD displays 64 characters on a 4 line screen. The high resolution provides an easy-to-read format from a wide viewing angle. The display also contains an electroluminescent panel for evenly distributed backlighting of the display.	Operators have a high-contrast, easy-to-read display that minimizes eye fatigue. The backlighting feature allows for operation in dark areas, common in many applications.
Optional direct-connect laser scanning	Popular non-contact laser scanners can be connected directly to the RT2210XL.	This option eliminates the need for costly and cumbersome external interfaces and power supplies.
Built-in diagnostic self-test	Built-in self-test is initiated any time the unit is powered up. The self test verifies proper operation of the microprocessor, memory and input/output circuits.	When the unit is initially powered-up, the self-test provides the operator with a high degree of confidence the unit is operating properly.
Supports several communications protocols	In addition to our own standard ASYNC protocol, the RT2210XL can support SNA/SDLC protocol.	Allows fast implementation and greater compatibility in IBM 3270 and System 36/38 environments.

NORAND® RT2210XL Accessories for Ease and Flexibility of Use

Norand offers accessories to support the RT2210XL Radio Data Terminal. Each accessory is designed to make the RT2210XL easier to implement and use. Select the accessories to help you in your special environment or situation.



The NC116 Quad Lockbox Charger

• Battery Chargers

Several battery charging options are available from Norand. You can rejuvenate the battery pack of your RT2210XL by using the NC120 Single Unit Charger, or the NC146 Quad Lockbox for multiple unit, simultaneous charge.

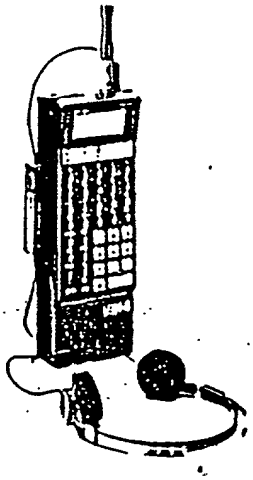
• Vehicle Mount

A rugged material handling vehicle mount secures your RT2210XL when in use on a forklift truck. The mount can be installed to provide optimum accessability for your operator and adjusts to various angles for easy keystroke entry. The terminal slides quickly and easily out of the vehicle mount for hand held terminal operation away from the forklift truck. A cradle on the side of the mount provides the sturdy base for supporting your scanner when not in use.

Vehicle Mount

• Audio Headphones

The headphones for the RT2210XL are engineered for use in obtrusive, ambient audio situations. An audio jack (next to the radio antenna mount) provides the interconnection for the headphones. Audible signals signify a properly or improperly decoded bar code symbol. The user is also alerted when new data is received from the host computer.



Audio Headphones

• Leather Carrying Cases

Durable leather carrying cases are available for the RT2210XL and a wide variety of scanners on the market. The heavy-grade, leather construction provides a practical accessory for your RT2210XL Radio Data Network.

The leather carrying case for the RT2210XL can be worn across the shoulder or attached to a belt or holster for user versatility. An adjustable lock on the holster allows you to position your hand-held terminal in a variety of angles (positions) for ease and accessability. Attachments for the holster are available for virtually any scanner or light pen.

Real-Time, On-Line Communications with the NORAND® RT2210XL and the Radio Data Network System

To fully appreciate the features and benefits of the RT2210XL, it is important to understand how the hand-held terminal integrates into the Radio Data Network. The RT2210XL operates under the control of your host computer. All terminal commands are initiated by the host computer, which communicates directly with a network communications multiplexer or controller.

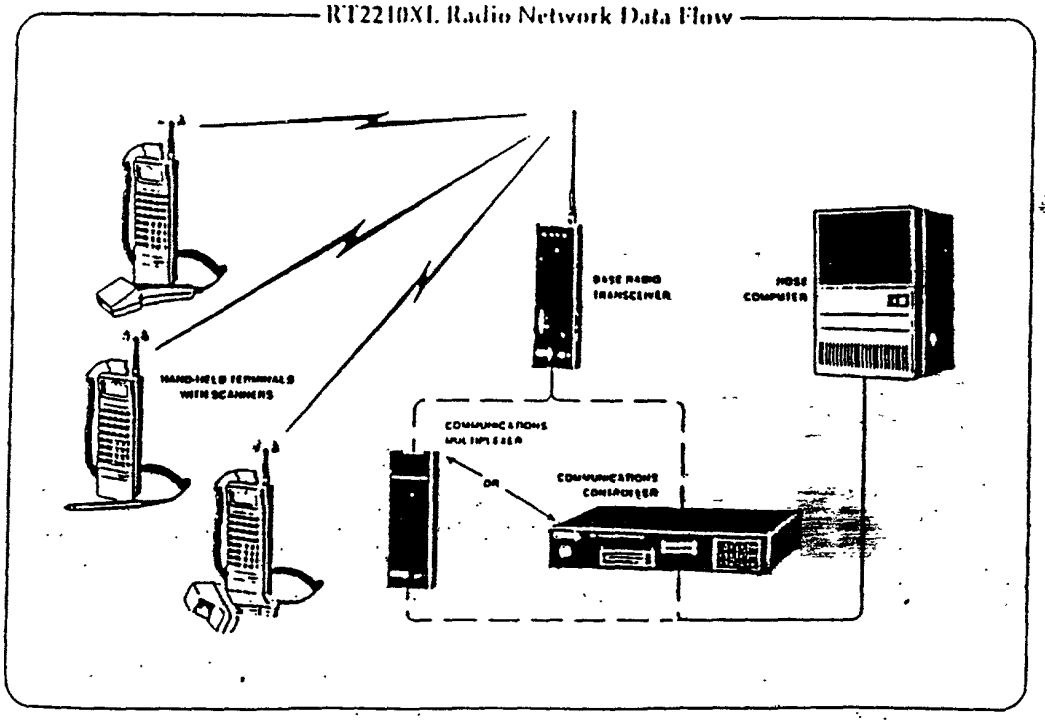
The multiplexer or controller handles the timing, protocol, and data buffering between the host computer and the RT2210XL Radio Data Terminals from Norand. When a terminal command is received from the host computer, the multiplexer (or controller) converts the command into the RT2210XL format, and transmits the command to the base radio transceiver. The base radio, utilizing the same high-performance radio used in the RT2210XL, transmits the commands on to the terminal.

The RT2210XL receives the host command from the base radio transceiver. This enables the use of either the terminal's keyboard, a bar code scanner, or both. Once the desired data has been entered, it is then transmitted back to the multiplexer (or controller) via the base radio transceiver. The multiplexer (or controller) then converts the information back into the host computer format and instantaneously transmits the data to the host computer.

All application software for the RT2210XL resides in the host computer and can be written in any programming language. The need for special development systems is eliminated, allowing for faster program implementation. In many cases, just a few simple command codes need to be added. This results in ease of system maintenance and allows you the freedom to upgrade.

Norand has radio networks available which utilize ASYNC and SNA/SDLC protocols including IBM® 3270 emulation and System 36/38 compatibility. This host programming flexibility translates to a fast, cost-effective implementation.

RT2210XL Radio Network Data Flow



RT2210XL Radio Data Terminal
SPECIFICATIONS

Product Features:

Transceiver: Incorporates a 2 watt (UHF) frequency modulated (FM) radio transceiver controlled by the microprocessor. Type accepted per FCC Rules & Regulations, Part 90, Private Land Mobile Radio Service

Liquid Crystal Display (LCD): 64-character, dot matrix LCD configured as 4 lines x 16 characters with four status indicators

Audio Alert: An audible buzzer is activated under host control

Annunciators: Four LCD annunciators indicate low battery, shift mode, radio transmitting, and radio receiving

Keyboard: 39-key (optional 20 key) tactile feel

Self Diagnostics: Self diagnostics performed on power up on all memories and input/output circuitry

Backlighting: LCD is backlit using an electroluminescent panel

Static Shock Protection: Terminal is hardened against electrostatic discharge up to 16,000 volts

Shielding: Conforms to FCC Part 15 for Class A computing devices

Interface: 15 pin D connector

Hand Strap: Elastic strap for back of terminal secures terminal firmly in hand

Device Features:

Microprocessor: A CMOS microprocessor has been selected for its processing ability and low power consumption

Network Address Switch: User can set the terminal's address by use of a rotary switch. Device is not used on terminals with non-volatile RAM

Non-Volatile RAM (optional): Provides data protection for the RAM buffer even when the terminal is turned off or the battery pack is removed

Physical Dimensions:

Size: 9.1"x 3.1" x 1.8" (LWH) (23.9cm x 8.6cm x 4.6cm)

Weight: 2 pounds (907g)

Environmental Characteristics:

Temperature:

Operating: 14° to 120°F (-10° to 50°C)

Storage: -4° to 140°F (-20° to 60°C)

Recharging: 40° to 104°F (5° to 40°C)

Humidity: 0 to 90% noncondensing

Altitude: To 10,000 feet (3048 meters) above sea level

Internal Power Source:

Battery Cells: Nickel cadmium batteries

Operating Time: Four 1-cell batteries - 10 hours typical for continuous receive mode without scanner and backlighting

RT2210XL Battery Pack Characteristics:

Normal Recharge: A recharge cycle is completed in 14 hours

Standby Holding Charge: Maintains the batteries at full charge by supplying a trickle charge rate

Low Battery Indicator: Visual annunciator indicating low battery is displayed

Charging Sources: AC adapter type battery charger or quad (1 unit) charger holders

Radio Characteristics:

Radiated Power: 2 watts

Frequency Range: 450 to 470 MHz and 896 to 940 MHz

Antenna: 2 inches (5 cm) stub

Data Rate: 9600 baud

Modulation: NRZ/FFSK

Bar Code Scanning Support:

NORAND 20/20 CCD Bar Code Scanners
Laser Scanners (HeNe and Laser Diode)
Pen Wands

Bar Code Symbologies Supported:
UPC/EAN, UPC/EAN with add-ons, Code 11, Code 39, Extended Code 39, Code 93, Code 128, Interleaved 2 of 5, Plessey, Codabar, ABC Codabar, Straight 2 of 5, and Computer Identics 2 of 5

NORAND®

Norand Corporation
550 Second Street S.E.
Cedar Rapids, Iowa 52401
Phone: 319/369-3156
1-800 553-5971 toll free (ext. 3156)

Norand Data Systems, Ltd.
951 Denison Street
Unit #4
Markham, Ontario
Canada L3R 3W9
Phone: 416-477-1818

Norand (U.K.) Ltd.
5 Bennet Court
Bennet Road
Reading, Berkshire RG2 0QX
England
Phone: (44) 734-861221

APPENDIX D2

Brochure entitled "RT3210 Radio
Data Terminal" (Four Sides)

NORAND
DATA SYSTEMS

RT3210
Radio Data Terminal

The Durability of the NORAND® RT3210 Provides the Flexibility to Fit the Way You Do Business

The new RT3210 Radio Frequency Terminal is a revolutionarily designed hand-held that provides the optimum in system flexibility and durability. It's the latest in an established line of hand-held terminals and computers offered from Norand Corporation.

The RT3210's industrialized, die-cast metal alloy housing makes it the perfect fit for most any application requiring extended use...even in the most rugged environments. Tempered to withstand extremes in drop, shock, vibration, and temperature, the RT3210 is designed to meet and exceed stringent Military Specifications 810D.

FEATURES:

- Industrialized metal alloy housing for optimum durability
- Ergonomically contoured for operational ease of use and comfort
- Unique interchangeable mobile/forklift mount design for maximized flexibility
- Industry-standard 9-pin scanner connector for both 5V and 12V scanning peripherals
- Printer port 15-pin connector supports popular bar code printers

The Versatility of the NORAND® RT3210 RF Terminal Offers Unsurpassed Value

The NORAND® RT3210 Hand-Held Terminal features the latest technological and ergonomic innovations. The solid RT3210 ensures dependable operations in the most adverse conditions.

The Radio Frequency Terminal's environmentally sound construction offers the optimum in system durability. The die-cast metal alloy housing is tested to withstand drops to a concrete surface of 4-feet on 6 sides. The NORAND RT3210 redefines reliability for all applications in any environment requiring on-line wireless data communications.

All application software for the RT3210 resides in the host computer and can be written in any programming language. The need for special development systems is eliminated, allowing for faster program implementation.

The RT3210 extends the boundaries of computerization through two-way, radio frequency communications. It goes where you go, allowing you to access and update information (stored in your host system) from remote areas of your facility.

The contoured styling of the RT3210 makes it the perfect companion of retail personnel for ensuring pricing accuracy, receiving and tracking inventory, setting ad promotions, and performing many other operational control functions.

• New Ergonomic Design

The contoured styling of the RT3210 conforms to the user's hand. This ergonomic styling makes extended use one of comfort and unconstrained operation.

A Super Twist LCD (liquid crystal display) provides crisp resolution in a user configurable 16 line by 21 character or 10 line by 16 character display. A contrast control adjustment feature enhances readibility in varying light situations. The display also incorporates an electroluminescent backlight for use in low light and nighttime operation.

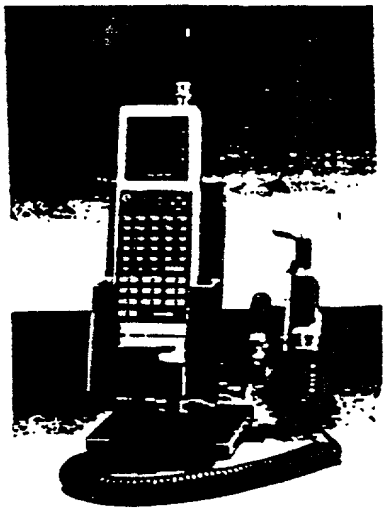
Special display features guide users through each step of operation. The terminal's status can be determined at a glance. Nine indicators at the bottom of the LCD convey the unit's state of operation. In addition, audible alerts inform the user of special conditions and provide audible response to key strokes.

Keyboard styling and function keys enhance the operational efficiency of the RT3210. The sealed 47-key alphanumeric keyboard is designed for optimum user acceptance. An integrated cursor control provides expanded mobility for rapid movement across the display screen.

The hand-held terminal is powered by a 7.5 volts dc (nominal), nickel-cadmium battery. The battery pack effortlessly slides in and out of the RT3210 for expedient battery interchangeability. A latching mechanism secures the battery in the RF terminal during operation.

• Modular Approach Maximizes Flexibility

The value of the RT3210 is enhanced with the introduction of the first interchangeable vehicle communications system. It can be forklift mounted or instantly removed for use as a portable hand-held terminal, independent of the forklift. This unique flexibility provides the modularity freedom to use the RT3210 in an extended range of applications and/or situations.



The NORAND® RT3210 Terminal slides in and out of the communications vehicle mount for mobile or hand-held flexibility.

Powered (charges [illegible] when engaged in the cradle [illegible] mount) and non-powered vehicle mounts are available for the RT3210 for use in industrial environments. The durable swivel mounts (tilt and rotation) securely hold the terminal in place and provide easy operator accessibility.

The environmentally sound RT3210 is designed to withstand harsh industrial conditions.

• Peripheral Connectivity for Expanded Versatility

The RT3210 is equipped with a 9-pin D-sub connector for interface to industry standard bar code scanning devices. It also supports both 5 volt and 12 volt helium-neon scanners in addition to laser diodes, CCD's, and light pens.

All major bar code symbologies are supported by the NORAND® RT3210 Hand-Held Terminal.

A printer output communications port serves as the junction for connecting the hand-held to bar code printers and other automatic identification peripherals. The 15-pin D-sub connector is located at the base of the RT3210 for easy accessibility and connectivity.

• Exclusive Automatic Baud-Rate Switching

The new third generation digital radio of the NORAND RT3210 delivers unmatched coverage for maximized throughput. The exclusive automatic baud-rate switching (pat. pending) on the RF link ensures the fastest reliable communication of data.

The variable rate capability of the terminal constantly monitors the link and transmits at 9,600 b.p.s. when conditions permit...or switches to 4,800 baud to boost transmission reliability in fringe coverage areas. A data compression feature and exclusive automatic baud-rate switching greatly enhance response time.

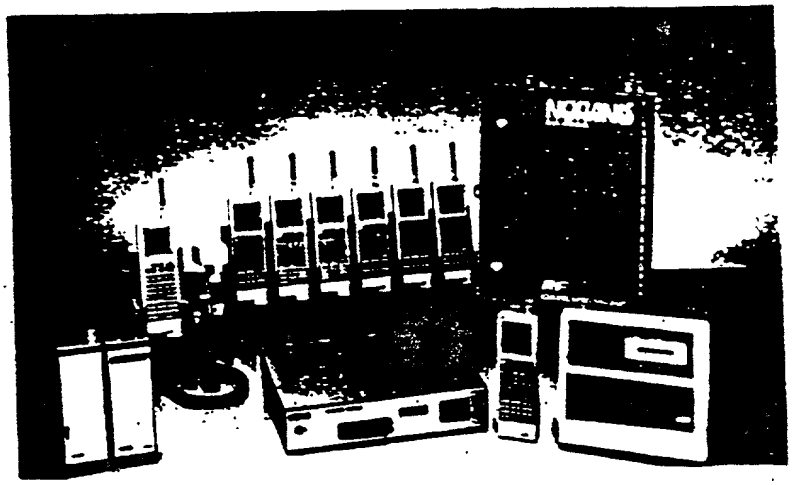
The RT3210 also incorporates the Norand exclusive Real Time Control system protocol. This unique RF protocol speeds response time and is exceptionally effective when numerous terminals are simultaneously in use.

• System Architecture

The RT3210 provides the link between your host computer and remote areas of your facility and operates under the control of the host computer. The multiplexer or controller handles the timing, protocol, and data buffering between the host and the hand-held terminals. The high-performance base radio transceiver transmits the commands to the terminal.

Because the RT3210 requires no special user programming, it quickly and easily integrates into any host computer system. Host system software can be written in the language of the user's choice. A few command code additions to existing software is generally all that is needed to get the Norand system up and running in your operation. Updates, changes, and additions to the software are made solely at the host system.

The simple, yet comprehensive RF system approach from Norand is unparalleled in the industry. This philosophy makes the RT3210 the most versatile hand-held terminal on the market.



The RT3210 hand-held terminals, base tranceivers, communicat... and controllers, vehicle mounts, and charging options offer u... to fit the way you do business.

RT3210 Radio Data Terminal
SPECIFICATIONS

Product Features:

Transceiver: Incorporates a 2 watt (UHF) frequency modulated (FM) radio transceiver controlled by the microprocessor. Type accepted per FCC Rules & Regulations, Parts 2 & 90, Private Land Mobile Radio Service

Liquid Crystal Display (LCD): Super Twist LCD with configurable 16 line x 21 character and 10 line x 16 character display feature (1 line of display designated for annunciators) with contrast control adjustment feature

Backlighting: LCD is backlit using an electroluminescent panel

Annunciators: TX (transmitting), RX (receiving), CL (communications loss), BATT (low battery), SHFT (shift), E (external power), ALT (alternate), FUNC (function), and + (9600 baud) are displayed on bottom line of LCD

Keyboard: Sealed elastomer 47-key alphanumeric tactile feel

Self-Diagnostics: Performed on power-up with built-in user accessible diagnostics

Audio Alert: An audible buzzer is activated under host control

Static Shock Protection: Terminal is hardened against electrostatic discharge up to 20,000 volts

Shielding: Conforms to FCC Part 15 for Class A computing devices

Printer Port: 15-Pin D-Sub connector

Scanner Interface: 9-Pin D-Sub connector with selectable 5 volt and 12 volt scanning options

Hand Strap: Elastic strap (on back of terminal) secures terminal firmly in hand to facilitate handling

RAM: 64K bytes x 8 bits, nonvolatile with lithium battery back-up

ROM: 64K bytes x 8 bits

Device Features:

Microprocessor: High performance CMOS (80C552)

Nonvolatile RAM: Provides data protection for the RAM buffer even when the terminal is turned off or the battery pack is removed

Environmental Standards: Designed to meet and exceed stringent Military Specifications 810D for shock, drop, vibration, and temperature. Undergoing certification process for Intrinsic Safety

Physical Dimensions:

Size: 10.25" x 3.35" x 2.25" (LWD)
(26.04cm x 8.51cm x 5.72cm)

Antenna Length: 3.25" (8.26cm)

Weight: 42 ounces (1.2kg)

Environmental Characteristics:

Temperature:

Operating: -4° to 122°F (-20° to 50°C)

Storage: -22° to 140°F (-30° to 60°C)

Recharging: 41° to 104°F (5°to 40°C)

Humidity: 0 to 90% noncondensing

Altitude: To 10,000 feet (3,048 meters) above sea level

Internal Power Source:

Battery Cells: Rapid charge nickel-cadmium batteries

Voltage: 7.5 VDC (nominal)

Operating Time From Batteries: 10 hours typical, based on customer usage

RT3210 Battery Pack Characteristics:

Normal Recharge: Recharge cycle complete in less than 8 hours

Fast Charge: Fast recharge cycle complete in less than 2 hours

Standby Holding Charge: Maintains the batteries at full charge by supplying a trickle charge rate

Low Battery Indicator: Visual annunciator (BAT) indicating low battery is displayed on bottom line of LCD

Battery Pack Charging:

Charging Sources: AC adapter-type single terminal chargers, multi-terminal chargers, and multi-battery pack chargers available

Input Power: 110/220 VAC, 50/60 Hz

Electrical Safety Approvals: UL, CSA

Radio Characteristics:

Radiated Power: 2 watts (maximum)

Frequency Range: 450 to 470 MHz

RF Data Rate: 4800 baud/9600 baud

Type Certification:

USA: FCC (Parts 2 & 90)
Canada: DOC (available 1990)

Bar Code Scanning Support:

CCD Bar Code Scanners
Laser Scanners (HeNe and Laser Diode)
Pen Wands

Bar Code Symbologies Supported:
UPC/EAN, UPC/EAN with add-ons, Code 39, Extended Code 39, Encoded Code 39, Code 93, Code 128, Interleaved 2 of 5, Plessey, Codabar, ABC Codabar, Straight 2 of 5, and Computer Identics 2 of 5

NORAND
DATA SYSTEMS

Norand Corporation
550 Second Street S.E.
Cedar Rapids, Iowa 52401
Phone: 319/369-3156
1-800-553-5971 toll free (ext. 3156)

Norand Data Systems Ltd.
951 Denison Street
Unit #4
Markham, Ontario
Canada L3R 3W9
Phone: 416-477-1818

Norand (U.K.) Ltd.
5 Bennet Court
Bennet Road
Reading, Berkshire RG2 0QX
England
Phone: (44) 734-861221

APPENDIX D3

Brochure entitled "RT3310 & RT3410
Radio Data Terminals (Four Sides)

NORAND®
DATA SYSTEMS

RT3310 & RT3410
Radio Data Terminals

More RF Functionality and Value through Third-Generation NORAND® Technology

The new 3000 Series RF terminals maximize your return on investment. The RT3310 and RT3410 Radio Data Terminals are compatible with all Norand® asynchronous system components. Their third-generation Norand engineering helps ensure a reliable wireless link to your host computer from anywhere in your facilities.

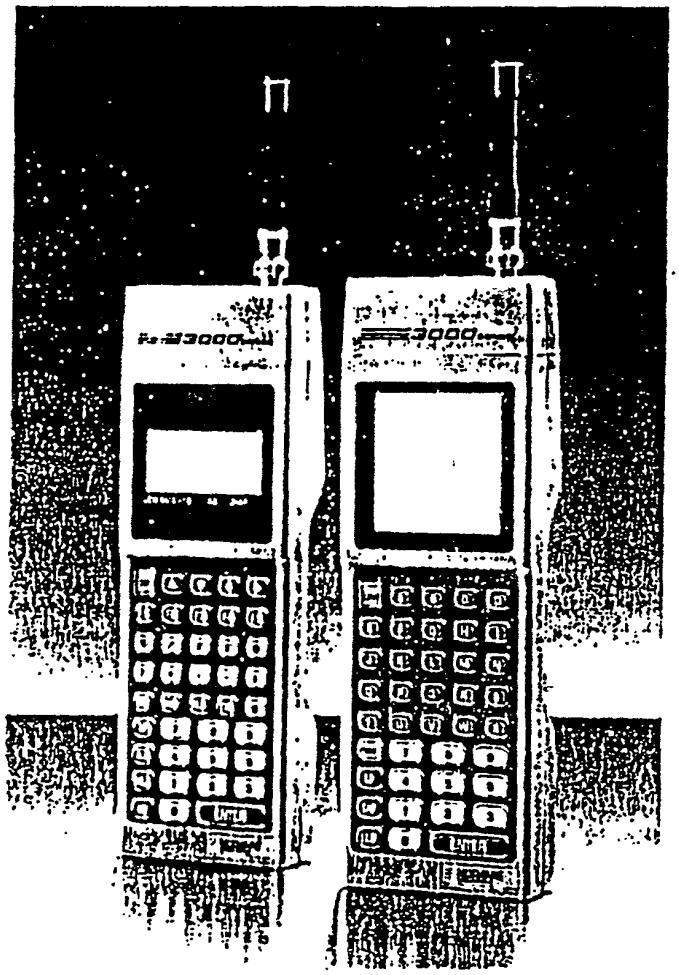
Working together with our new RM3216 multiplexer, they establish a new standard in performance, thanks to their unmatched response time. Employees can spend more time working, less time waiting for the communication of transactions.

These new terminals also set the standard in user comfort and flexibility due to their new ergonomic shape and increased connectivity.

- **The RT3310 and RT3410 Redefine Performance in RF Terminals**

RF extends the boundaries of computerization through reliable two-way radio communication. It gives you immediate access to the information in your host computer from anywhere in your facilities – and the ability to update that information in seconds.

The RT3310 and RT3410 terminals make RF even better. They give you more functionality... more versatility... and they're easier and more comfortable to use. A product of third-generation Norand technology, they're the best terminals anyone has ever offered.



FEATURES:

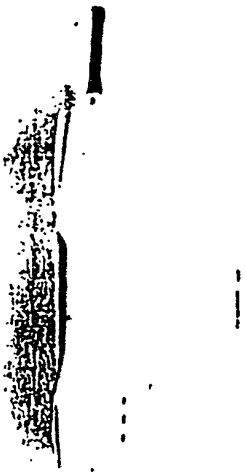
- SureGrip Design™ for comfort
- Exclusive Adaptive Polled Protocol™ for rapid response time
- Exclusive automatic baud-rate switching
- 9- and 15-pin connectors for scanners and printers

• Easier to Hold and Use

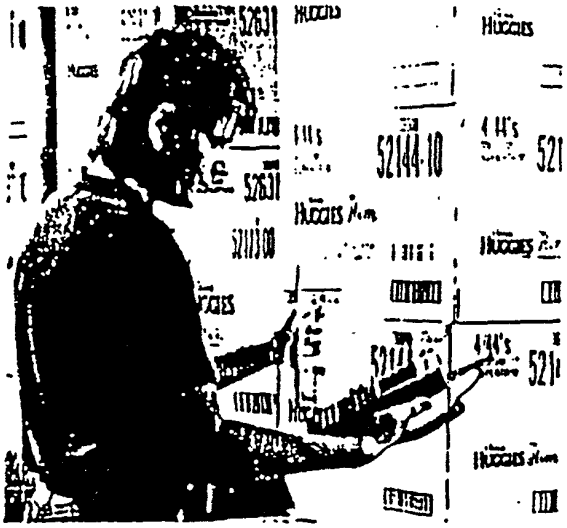
The 3000 Series Terminals feature our SureGrip Design™ for greater handling care. The handsome, contoured shape conforms to the natural position of the hand. A hand strap offers firm, yet gentle support to secure holding the terminal.

Special display features guide users through each step of operation. Your employees merely respond to the prompts that appear on the screen.

Both terminals can be ordered with a standard 39-key color-coded, alphanumeric keyboard or a color-coded retail application keyboard. The retail application keyboard offers five user-definable keys to simplify performing complex functions. They speed the work process by reducing the number of required keystrokes.



The SureGrip Design™ of the 3000 Series Radio Data Terminals contours to the hand of the user for comfort and unconstrained operation.



NORAND® RT provides on-line access of information residing in your host system. On-line access speeds the transactional process and enhances the accuracy of the data collection process.

• Choose a 4- or 16-Line Display

The RT3110 comes with a 4 line by 16 character display. The RT3110 offers a 16 line by 21 character display for detailed applications. The 16 line by 21 character display can also be configured to a 10 line by 16 character display for enhanced user legibility. A display package, that can upgrade your RT3110 to an RT3110s available through the NORAND® Service Center.

The easy-to-read Super Twist liquid crystal displays (LCD) feature superior readability in varying light conditions. The display also incorporates an electroluminescent backlight for use in low light and nighttime operation.

• More Worker Productivity

Operating in harmony with the NORAND RM3216 multiplexer, the terminals speed work. Our exclusive Enhanced Adaptive Polled Protocol™ reduces response time through faster polling techniques and universal addressing.

Support for the RT3410's 16-line display reduces the time required to communicate data for the display.

• Exclusive Automatic Baud-Rate Switching

Our patented automatic baud-rate switching also improves worker productivity by ensuring the fastest reliable communication of data.

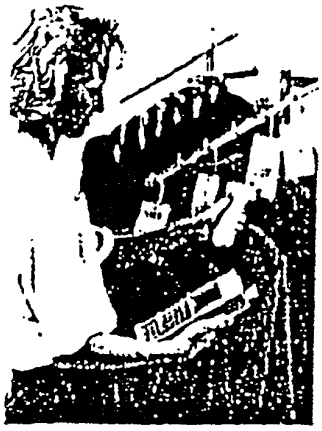
The variable rate capability of the terminal constantly monitors the RF link and transmits at 9,600 b.p.s. when conditions permit, or switches to 4,800 b.p.s. to boost the reliability of transmissions in fringe coverage areas.

Our third generation digital radio module delivers maximum coverage.

• Greater Flexibility through Expanded Peripheral Connectivity

The RT3310 and RT3410 are equipped with 9- and 15 pin D-sub connectors to add to their versatility.

The industry standard 9-pin bar code scanner connector supports a variety of 5-volt scanners such as laser diodes, CCDs, and light pens. These terminals support all major bar code symbologies.



Ensuring pricing accuracy in retail environments is one of the many applications employed with the implementation of the 3000 Series Radio Data Systems.

The 15 pin D-sub connector allows the connection of bar code printers and other automatic identification peripherals.

It allows the connection of the new NP1800 printer for creating bar codes in remote areas. Completely portable with its own power source, the NP1800 will print shelf labels or tag stock wherever it is needed.

• Reliable Rechargeable Battery Power

Both terminals are powered by a 7.5 volt DC (nominal) nickel-cadmium battery. The battery pack slides in and out of the terminal easily for fast battery replacement.

A latching mechanism secures the battery in the terminal during operation.

• A Simpler Architecture

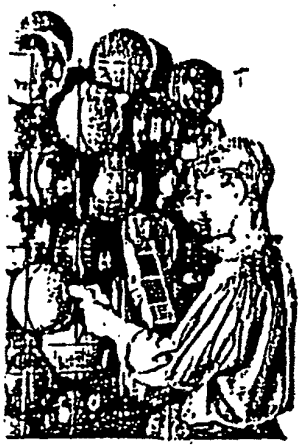
The advantages of the RT3310 and RT3410 lie within their unique system architecture. All software resides in your host computer and can be written in any programming language.

The need for special development systems is eliminated, allowing faster program implementation. A few command code additions to existing software is generally all that is needed to get the system up and running.

This simple approach helps improve the reliability of the terminal because it eliminates unnecessary replacement ROM chips. It also allows easier program updates, changes, and additions. Instead of updating each terminal in your RF network, you merely update the host software.

The RT3310 and RT3410 terminals are part of the advanced 3000 Series RF network systems from Norand. The RM3216 multiplexer handles timing, protocol, and data buffering between the host and the hand held terminals. And the RB2212 high-performance base radio transceiver facilitates reliable radio communication between the host and the terminals.

A number of other components and accessories complete making the most of RF for you.



The 3000 Series Radio Data System easily integrates into any host computer system. A few command code additions to existing software is generally all that is needed to get the Norand® system up and running in your operation.

RT3310 & RT3410 Radio Data Terminals
SPECIFICATIONS

Product Features:

Transceiver: Incorporates a 2 watt (UHF) frequency modulated (FM) radio transceiver controlled by the microprocessor. Type accepted per FCC Rules & Regulations, Parts 2 & 90, Private Land Mobile Radio Service

RT3310 Display: Liquid crystal display (LCD) 4 line by 16 character with backlighting and 4 annunciators (low battery, shift mode, radio transmitting, radio receiving). A display package, available through the NORAND® Service Center, can upgrade the RT3310 to an RT3410

RT3410 Display: Super Twist 128 x 128 pixel LCD with configurable 16 line x 21 character and 10 line x 16 character display feature and electroluminescent backlighting. Semi-temperature compensating contrast control and manual contrast adjustment.

Keyboard: Sealed elastomer 39 key standard color-coded alphanumeric or color-coded with retail symbols

Self Diagnostics: Performed on power up with built in user accessible diagnostics

Audio Alert: An audible buzzer is activated under host control

Electrostatic Discharge Protection: Terminals hardened against electrostatic discharge up to 20,000 volts

Shielding: Conforms to FCC Part 15 for Class A computing devices

RS 232 Support: A 15 pin D sub connector allows connection to a variety of peripherals such as bar code readers or other data collection devices

Scanner Interface: Industry standard 9 pin D sub connector with 5 volt scanning options

Hand Strap: Elastic strap (on back of terminal) secures terminal firmly in hand to facilitate handling

RAM: 64K bytes x 8 bits, nonvolatile with lithium battery back-up

ROM: 64K bytes x 8 bits

Device Features:

Microprocessor: High performance CMOS (80C552)

Nonvolatile-RAM: Provides data protection for the RAM buffer even when the terminal is turned off or the battery pack is removed

Physical Dimensions:

Size: 9.6" x 3.5" x 1.9" (3.3V10)
(24.38cm x 8.89cm x 4.83cm)

Weight: 33 ounces (938 g)
with battery pack

Environmental Characteristics:

Temperature:

Operating: 20° to 110°F
(-6° to 43°C)

Normal Battery Charging: 41° to 104°F
(5° to 40°C)

Storage: -22° to 140°F (-30° to 60°C)

Humidity: 10 to 90% noncondensing

Altitude: To 10,000 feet (3048 meters) above sea level

Internal Power Source:

Battery Cells: Nickel cadmium batteries

Voltage: 7.5 VDC (nominal)

Operating Time From Batteries: 8 - 10 hours typical depending on customer usage

Battery Pack Characteristics:

Normal Recharge: Recharge cycle complete in 12 hours

Standby Holding Charge: Maintains the batteries at full charge by supplying a trickle charge rate

Low Battery Indicator: Visual annunciator (LED indicating low battery to display on bottom-loaded LCD)

Battery Pack Charging:

Charging Sources: AC adapter type single terminal chargers, multi terminal chargers, and multi battery pack chargers available

Input Power: 110/220 VAC, 50/60 Hz

Electrical Safety Approvals: UL, CSA

Radio Characteristics:

Radiated Power: 2 watts (maximum)

Frequency Range: 450 to 470 MHz

RF Data Rate: 4800 baud/9600 baud automatic baud rate switching dependent upon compatible system configuration)

Type Certification:

USA: FCC (Parts 2 & 90)
Canada: DOC (available 1991)

Bar Code Scanning Support:

CCD Bar Code Scanners (5 volt)
Laser Scanners (5 volt)
Light Pens (5 volt)

Electrical Interface: Incorporates a 9-pin male, captive type D-sub connector on top end of the terminals. Interface cable available for connectivity to scanners with NORAND 15-pin D-sub connectors.

Bar Code Symbologies Support: UPC/EAN, UPC/EAN with add-ons, Code 39, Extended Code 39, Encoded Code 39, Code 93, Code 128, Interleaved 2 of 5, Plessey, Codabar, ABC Codabar, Straight 2 of 5, and Computer Identics 2 of 5

NORAND®
DATA SYSTEMS

Norand Corporation
550 Second Street S.E.
Cedar Rapids, Iowa 52401
Phone: 319-369-3156
1-800 553-5971 toll free (ext. 3156)

Norand Data Systems, Ltd.
951 Denison Street
Unit #1
Markham, Ontario
Canada L3R 3W9
Phone: 416 477-1818

Norand (U.K.) Ltd.
5 Bennet Court
Bennet Road
Reading, Berkshire RG2 0QX
England
Phone: (44) 734-861721

The goal of Norand is
100% customer satisfaction.
Customer Satisfaction Hot Line:
1-800-221-9236

940 329 010 Printed in U.S.A.

In a continuing effort to improve our products, Norand Corporation reserves the right to change specifications and features without prior notice.

APPENDIX D4

Brochure entitled "RT1000 Radio
Data Terminal" (Two Sides)

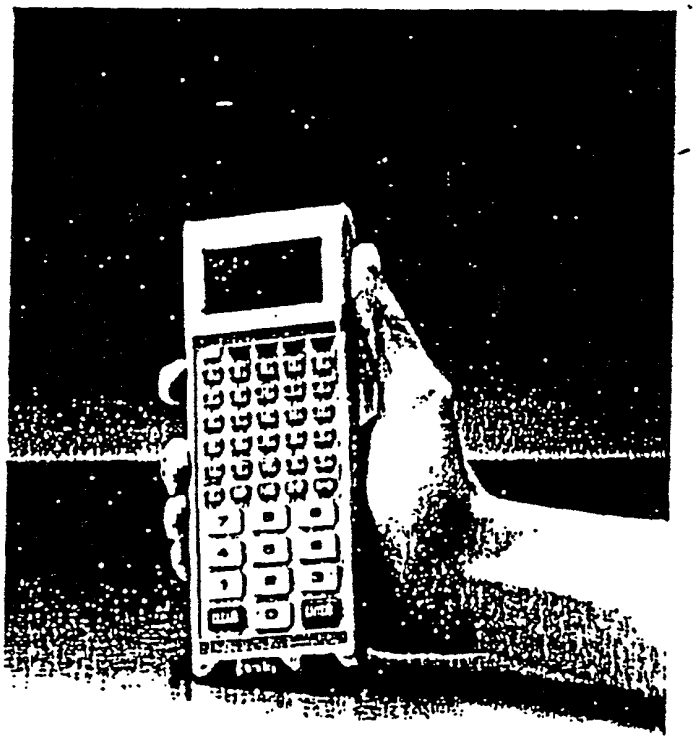
NORAND®
DATA SYSTEMS

RT1000
Radio Data Terminal

Introducing the NORAND® RT1000 Radio Data Terminal, Your Lightweight On-Line Productivity Tool that Fits in the Palm of Your Hand

Norand Corporation announces the newest addition to its family of products... the RT1000 Radio Data Terminal. This hand-held radio terminal is packed with the features you've been asking for and offers the advantages of on-line, real-time communications at a price guaranteed to maximize your company's return on investment.

This radio frequency (RF) terminal is so compact and lightweight it can fit into your shirt pocket. No other terminal comes close to matching the value offered by the RT1000 Radio Data Terminal.



The RT1000 Terminal features a 47-key alphanumeric keyboard with 4 soft-programmable keys, defined by the application software and downloaded when the terminal is initialized. A shift mode on the keyboard allows selection for up to 8 programmable characters along with upper case and lower case support. This keyboard layout makes it ideal for domestic and international applications requiring unique character sets.

The terminal incorporates a 4-line by 16-character liquid crystal display. The large characters on the Reflective Super Twist display enhance readability, even in low light environments.

The innovative architecture of the RT1000 Radio Data Terminal includes a central processor unit and analog board incorporating surface-mount, memory-on-chip technology. In addition, a patented self-contained, removable radio module gives the user the flexibility to change operating frequency on the spot. This unique modular design makes the RT1000 one of the most versatile, reliable, and serviceable products on the market.

The RT1000 is the ideal choice for new and current NORAND® RF users who are looking for a compact, lightweight, attractively priced solution to their data collection process. The backward and forward compatibility of the RT1000 Radio Data Terminal makes future upgrades quick and easy to implement. This insures the value of your investment now and for years to come.

FEATURES:

- Simple to install and use
- Interchangeable radio module
- Compatible with all current production Norand® products
- Compact functionality for maximized value

RT1000 Radio Data Terminal
SPECIFICATIONS

Product Features:

Transceiver: Incorporates a 1 watt (UHF) frequency modulated (FM) radio transceiver controlled by the microprocessor. Type accepted per FCC Rules & Regulations, Parts 2 & 90, Private Land Mobile Radio Service

Liquid Crystal Display (LCD): 4 line x 16 character Reflective Super Twist LCD

Keyboard: Elastomer 47 key alpha-numeric with 4 soft-programmable character keys (8 special characters possible with shift key engaged)

Annunciators: Battery recharge indicator on the terminal's LCD

Radio Module: Patented self contained, interchangeable 1 channel radio module with built-in receiver self-test

Radio Antenna: Stud-mount, screw on antenna connects directly to the end of the radio module

Drop Survival: Designed to withstand 1 foot drop to concrete

Hand Strap: Incorporates a user replaceable, elastic hand strap to secure the terminal firmly in hand

Belt Attachment Point: Removable clip allows terminal to be fastened to the belt

Device Features:

Central Processor Unit: 16-bit microcontroller

Shielding: Conforms to FCC Part 15 for Class A computing devices

Audible Tone: Audible annunciator to alert operator of action

Scanner Interface: 9-pin D subminiature connector for interface to 5 volt scanning peripherals with built-in scanning self-test

Electrostatic Discharge: Designed to withstand up to 20KV for Class C products

RAM: 512 bytes x 8 bits

ROM: 16K bytes x 8 bits (masked)

Physical Dimensions:

Size: 6.875" x 2.625" x 1.25" (LWD) (17.8cm x 6.68cm x 3.18cm)

Antenna Length: 2" (5.08cm)

Weight: 14.25 ounces (403g)

Environmental Characteristics:

Temperature

Operating: 32° to 122°F (0° to 50°C)

Storage: -22° to 158°F (-30° to 70°C)

Recharging: 41° to 104°F (5° to 40°C)

Humidity: 10 to 90% noncondensing

Altitude: To 10,000 feet (3,000 meters) above sea level

Internal Power Source

Battery Cells: Standard rechargeable nickel cadmium battery pack

Voltage: 7.2 VDC (nominal)

Operating Time Between Charges: 8 hours typical, based on customer usage of 8 scans/transmissions per minute

RT1000 Battery Pack Characteristics:

Normal Recharge: Complete in less than 8 hours

Pack Life: At least 500 discharge/charge cycles

Low Battery Indicator: Visual annunciator indicating low battery is displayed on the LCD

Battery Pack Charging:

Charging Sources: AC adapter-type single terminal chargers and multi-battery pack chargers available

Radio Characteristics:

Radiated Power: 1 watt (maximum)

Frequency Range: 450 to 470 MHz

RF Data Rate: 4800 baud

Bar Code Scanning Support:

CCD (5V)
Visible Laser Diode (5V)
Pen Wand (5V)

Bar Code Symbologies Supported: UPC, UPC with add-ons, EAN, EAN with add-ons, Code 39, Interleaved 2 of 5, Code 128, Plessey

NORAND
DATA SYSTEMS

Norand Corporation
550 Second Street S.E.
Cedar Rapids, Iowa 52401
Phone: 319 369 3100
1 800 553-5971 toll free (ext. 3156)

Norand Data Systems, Ltd.
951 Denison Street
Unit #1
Markham, Ontario
Canada L3R 1B9
Phone: 416 477-1818

Norand (U.K.) Ltd.
5 Bennet Court
Bennet Road
Reading, Berkshire RG2 0QX
England
Phone: (0) 734 864221

APPENDIX D5

Brochure entitled "RT5910 Mobile Mount Radio Terminal (Two Sides).

The NORAND® RT5910 Mobile Mount Terminal for Challenging Industrial Environments

RT5910 Mobile Mount Radio Terminal

The RT5910 Mobile Mount Terminal is the newest member of the NORAND® Radio Frequency (RF) Vehicle Communications System. The RT5910 Terminal's rugged design was developed for the harshest forklift-mount applications.

The Super Twist Transflective liquid crystal display (LCD) of the RT5910 has graphic display capabilities for enhanced readability and versatility. The LCD is backlit using long-life fiber optic technology.

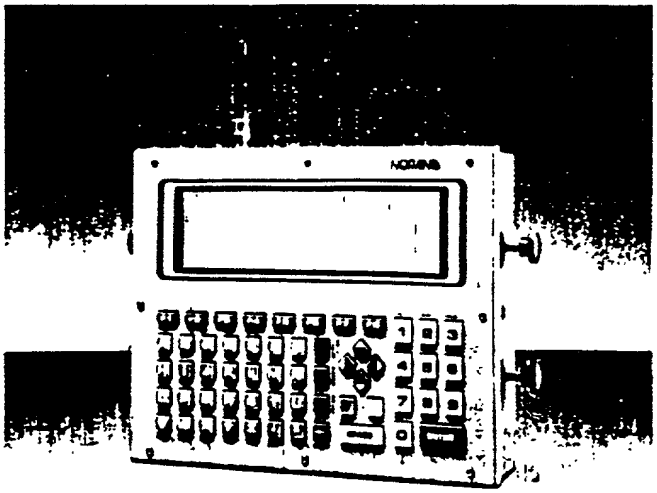
The display screen formatting of the RT5910 is compatible with the screen formats used in the RT3210 Radio Data Terminal and RD3990 Remote Display. The achievement of this application transparency allows the duplication of display screens without the need for host system software modifications.

The RT5910 also provides 3270, 5250, 7527, and VT220 (TCP/IP connectivity) emulation. This makes the RT5910 easy to integrate into your existing system and gives you the freedom to work directly with host data from remote areas of your facility.

The RT5910 Terminal is designed to meet NEMA 3 standards and incorporates a sealed elastomer keyboard. This 58-key alphanumeric keyboard has 24 functions keys to simplify and speed the data entry process. The over-sized keys are ideal for the large-handed or gloved user.

The Mobile Mount Radio Terminal supports 5 volt scanning. Also, a 15-pin RS232 port allows connection to a variety of peripherals such as bar code printers and other data collection devices.

The innovative architecture of the RT5910 requires no special user programming and integrates quickly and easily into most host computer systems. A few command code additions to existing software is generally all that is needed to get the system up and running in your operation. This simple, yet comprehensive RF system approach from Norand is unparalleled in the industry.



SYSTEM FEATURES

- Long-life Super Twist Transflective 16-line x 80-character liquid crystal display with graphic display capabilities
- Multiple screen format options for 8, 10, 12, 16, 21, and 25 line displays
- Sealed elastomer 58-key alphanumeric with 24 function keyboard
- Peripheral port (sealed) 15-pin male D-subminiature for connection to RS232 devices
- Scanner interface (sealed) 9-pin D-subminiature connector for 5 volt scanning
- Rugged case enclosure built to NEMA 3 standards for harsh environments

NORAND®
DATA SYSTEMS

RT5910 Mobile Mount Radio Terminal

SPECIFICATIONS

PRODUCT FEATURES

Transceiver: Incorporates a 2 watt (UHF) frequency modulated (FM) radio transceiver controlled by a microprocessor. Type accepted per FCC Rules & Regulations, Parts 2 & 90 Private Land Mobile Radio Service

Liquid Crystal Display (LCD): Super Twist Transflective LCD with configurable 16-line x 80-character display (includes positionable terminal status annunciators) with contrast control adjustment feature

Backlighting: LCD is backlit using fiber optic technology

Annunciators: TX (transmitting), RX (receiving), CL (communications loss), ALT (alternate), FUNC (function), and + (9600 baud), are displayed on the bottom line of the LCD

Keyboard: Sealed elastomer 58-key alphanumeric tactile feel with 24 function keys. ABC & QWERTY options

Audio Alert: An audible buzzer which is volume controlled via the keyboard

Self-Diagnostics: Performed on power-up with built-in user accessible diagnostics

Static Shock Protection: RT5910 Mobile Mount Radio is hardened against electrostatic discharge up to 20,000 volts

hielding: Conforms to FCC Part 15 for Class A computing devices

S232 Support: A sealed 15-pin male D-subminiature connector allows connection to a variety of peripherals such as bar code printers and other data collection devices. Supports data rates of 1200, 2400, 4800, 9600, and 19,200 bps

Terminal Emulation: 3270, 5250, 7527, VT220 (TCP/IP connectivity)

Scanner Port: A sealed 9-pin D-subminiature connector with 5 volt scanning capability

Power Input Connector: A sealed 2-pin circular locking connector for power connection

Power Conversion: Converts up to 72VDC forklift battery to 12VDC operating voltage

EPROM: 64K

Flash ROM: 128K

RAM: 128K (optional loadings 128K to 768K RAM)

PHYSICAL CHARACTERISTICS

Terminal Size: 15" x 10.2" x 4" (LWD)
(38.1cm x 25.9cm x 10.2cm)

Antenna Length: 3.25 inches (8.25cm)

Bracket Size: 13.68" x 11.8" x 7" (LWD)
(34.7cm x 30cm x 17.8cm)

Weight: 13.75 pounds (6.24kg) with bracket
9.75 pounds (4.42kg) without bracket

ENVIRONMENTAL CHARACTERISTICS

Operating Temperature: -22° to 122°F (-30° to 50°C)

Storage Temperature: -40° to 158°F (-40° to 70° C)

Humidity: 0 to 95% noncondensing

Standards: Designed to meet UL, CSA, and NEMA 3 standards and MIL-STD-810D

BAR CODE SCANNING SUPPORT

Bar Code Symbologies Supported: UPC, UPC with add-ons, EAN, EAN with add-ons, Code 128, Plessey, Code 3 of 9, Interleaved 2 of 5, Straight 2 of 5, Extended Code 3 of 9, Encoded 3 of 9

The goal of Norand is 100% customer satisfaction.
Customer Satisfaction Hot Line: 1-800-221-9236

NORAND®
DATA SYSTEMS

Norand Corporation
550 Second Street S.E.
Cedar Rapids, Iowa 52401
Phone: 319-369-3156
1-800-553-5971 toll free (ext. 3156)

Norand International Corporation
and Norand (U.K.) Limited
5 Bennet Court, Bennet Road
Reading, Berkshire RG2 0QX
England
Phone: (44) 734-861221
FAX: (44) 734-861156

Norand Data Systems, Ltd.
85 Citizen Court, Unit #1
Markham, Ontario,
Canada L6G 1A8
Phone: 416-477-1818
1-800-663-6157 toll free
FAX: 416-477-2242

APPENDIX E

Program Listing Showing
Preferred Control Instruction
for the Network Controller 40

```
/*****************************************************************
****

Copyright (C)1991, Norand Corporation.  All Rights Reserved.


Revision History:

** $Log:   N:/3250/rtc/src/callsign.c_v  $
 *
 *    Rev 1.6   27 Feb 1992 16:05:46   SOJKA
 * Dual Host Support Release and RS-422 Bug
 *
 *    Rev 1.5   18 Feb 1992 10:47:24   SOJKA
 * UIF cosmetic fixes V0.23
 *
 *    Rev 1.4   31 Dec 1991 11:22:36   sojka
 * Complied with Library V0.21.
 *
 *    Rev 1.3   31 Dec 1991 10:40:28   sojka
 * Complied with Library V0.21.
 *
 *    Rev 1.2   20 Nov 1991 20:09:44   sojka
 * Added Copyright Information and PVCS Log keyword.

*****************************************************************
****/



#include "main.h"

extern BYTE rad_tx_buffer[];
extern void (*rad_tx_routine)();
extern WORD rad_timer;
extern void (*rad_timer_routine)();

BYTE   callsign_data[128];
WORD   callsign_table[] =
{
        0x3D99,0x0000,0x0000,0x0000,0x0000,0x0000,0x0000,0x0000,
        0x0000,0x0000,0x0000,0x0000,0x0000,0x0000,0x0000,0x0000,
        0x0000,0x0000,0x0000,0x0000,0x0000,0x0000,0x0000,0x0000,
        0x0000,0x0000,0x0000,0x0000,0x0000,0x0000,0x0000,0x0000,
        0x0000,0x0000,0x0000,0x0000,0x0000,0x0000,0x0000,0x0000,
        0x09A6,0x09A6,0x0000,0x0000,0x0A5A,0x0000,0x0999,0x0196,
        0x02AA,0x02A9,0x02A5,0x029E,0x0255,0x0155,0x0156,0x015A,
        0x016A,0x01AA,0x056A,0x0666,0x0000,0x0000,0x0000,0x05A5,
        0x0000,0x0009,0x0056,0x0066,0x0016,0x0001,0x0065,0x001A,
        0x0055,0x0005,0x00A9,0x0026,0x0059,0x000A,0x0006,0x002A,
        0x0069,0x009A,0x0019,0x0015,0x0002,0x0025,0x0095,0x0029,
        0x0096,0x00A6,0x005A,0x0000,0x0000,0x0000,0x0000,0x0000,
        0x0000,0x0000,0x0000,0x0000,0x0000,0x0000,0x0000,0x0000,
        0x0000,0x0000,0x0000,0x0000,0x0000,0x0000,0x0000,0x0000,
        0x0000;0x0000,0x0000,0x0000,0x0000,0x0000,0x0000,0x0000,
```

```
        0x0000,0x0000,0x0000,0x0000,0x0000,0x0000,0x0000,0x0000
};

extern WORD rad_callsign_timer;
#define TWENTY_MINUTES                  120

#define CALLSIGN_END                    0
#define CALLSIGN_ON                             1
#define CALLSIGN_OFF                    2

#define CALLSIGN_INCREMENT              80
#define CALLSIGN_BYTES                  40

WORD callsign_ptr;

void callsign_set(x)
BYTE *x;
{

        BYTE *temp;
        WORD i;

        callsign_ptr = 0;
        rad_callsign_timer = 0;

        temp = &callsign_data[0];

        while (*x)
        {
                i = callsign_table[*x++];
                while (i)
                {
                        switch (i & 3)
                        {
                                case 1:
                                        *temp++=CALLSIGN_ON;
                                        break;
                                case 2:
                                        *temp++=CALLSIGN_ON;
                                        *temp++=CALLSIGN_ON;
                                        *temp++=CALLSIGN_ON;
                                        break;
                                default:
                                        break;
                        }
                        *temp++=CALLSIGN_OFF;
                        i >>= 2;
                }
                *temp++=CALLSIGN_OFF;
                *temp++=CALLSIGN_OFF;
        }
        *temp = CALLSIGN_END;
}

void callsign_next()
{
```

```
        switch (callsign_data[callsign_ptr])
        {
                case CALLSIGN_END:
                        rad_timer = 0;
                        post_mailbox(0xFFFF);
                        break;

                case CALLSIGN_ON:
                        rad_timer = 0;
                        rad_tx_routine = callsign_next;
                        rtc_tx_init(port);
                        break;

                case CALLSIGN_OFF:
                        rad_timer = CALLSIGN_INCREMENT;
                        rad_timer_routine = callsign_next;
                        break;
        }
        callsign_ptr++;
}

void callsign_check()
{

        if (!rad_callsign_timer)
        {
                lcd_tx_status(LCD_TX_CALLSIGN);
                if (callsign_data[0] != CALLSIGN_END)
                {
                        callsign_ptr = 0;

                        port = PORT0;
                        /* set up transmit */
                        ptt(port);
                        memset(&rad_tx_buffer,0x1c,CALLSIGN_BYTES);
                        rad_tx_length = CALLSIGN_BYTES;

                        callsign_next();

                        /* wait to be finished */
                        pend_mailbox(0);

                        ptt_off(port);
                }
                rad_callsign_timer = TWENTY_MINUTES;
        }
}
```

```
/*****************************************************************************
****

Copyright (C)1991, Norand Corporation.  All Rights Reserved.


Revision History:

** $Log:   N:/3250/rtc/src/crc.c_v  $
 *
 *    Rev 1.6   27 Feb 1992 16:05:56   SOJKA
 * Dual Host Support Release and RS-422 Bug
 *
 *    Rev 1.5   18 Feb 1992 10:47:36   SOJKA
 * UIF cosmetic fixes V0.23
 *
 *    Rev 1.4   31 Dec 1991 11:22:48   sojka
 * Complied with Library V0.21.
 *
 *    Rev 1.3   31 Dec 1991 10:40:38   sojka
 * Complied with Library V0.21.
 *
 *    Rev 1.2   20 Nov 1991 18:37:32   sojka
 * Added Copyright Information and PVCS Log keyword.

*****************************************************************************
****/



#include "main.h"

WORD crc_table[256] =
{
0x0000, 0xc0c1, 0xc181, 0x0140, 0xc301, 0x03c0, 0x0280, 0xc241,
0xc601, 0x06c0, 0x0780, 0xc741, 0x0500, 0xc5c1, 0xc481, 0x0440,
0xcc01, 0x0cc0, 0x0d80, 0xcd41, 0x0f00, 0xcfc1, 0xce81, 0x0e40,
0x0a00, 0xcac1, 0xcb81, 0x0b40, 0xc901, 0x09c0, 0x0880, 0xc841,
0xd801, 0x18c0, 0x1980, 0xd941, 0x1b00, 0xdbc1, 0xda81, 0x1a40,
0x1e00, 0xdec1, 0xdf81, 0x1f40, 0xdd01, 0x1dc0, 0x1c80, 0xdc41,
0x1400, 0xd4c1, 0xd581, 0x1540, 0xd701, 0x17c0, 0x1680, 0xd641,
0xd201, 0x12c0, 0x1380, 0xd341, 0x1100, 0xd1c1, 0xd081, 0x1040,
0xf001, 0x30c0, 0x3180, 0xf141, 0x3300, 0xf3c1, 0xf281, 0x3240,
0x3600, 0xf6c1, 0xf781, 0x3740, 0xf501, 0x35c0, 0x3480, 0xf441,
0x3c00, 0xfcc1, 0xfd81, 0x3d40, 0xff01, 0x3fc0, 0x3e80, 0xfe41,
0xfa01, 0x3ac0, 0x3b80, 0xfb41, 0x3900, 0xf9c1, 0xf881, 0x3840,
0x2800, 0xe8c1, 0xe981, 0x2940, 0xeb01, 0x2bc0, 0x2a80, 0xea41,
0xee01, 0x2ec0, 0x2f80, 0xef41, 0x2d00, 0xedc1, 0xec81, 0x2c40,
0xe401, 0x24c0, 0x2580, 0xe541, 0x2700, 0xe7c1, 0xe681, 0x2640,
0x2200, 0xe2c1, 0xe381, 0x2340, 0xe101, 0x21c0, 0x2080, 0xe041,
0xa001, 0x60c0, 0x6180, 0xa141, 0x6300, 0xa3c1, 0xa281, 0x6240,
0x6600, 0xa6c1, 0xa781, 0x6740, 0xa501, 0x65c0, 0x6480, 0xa441,
0x6c00, 0xacc1, 0xad81, 0x6d40, 0xaf01, 0x6fc0, 0x6e80, 0xae41,
0xaa01, 0x6ac0, 0x6b80, 0xab41, 0x6900, 0xa9c1, 0xa881, 0x6840,
0x7800, 0xb8c1, 0xb981, 0x7940, 0xbb01, 0x7bc0, 0x7a80, 0xba41,
```

```
0xbe01, 0x7ec0, 0x7f80, 0xbf41, 0x7d00, 0xbdc1, 0xbc81, 0x7c40,
0xb401, 0x74c0, 0x7580, 0xb541, 0x7700, 0xb7c1, 0xb681, 0x7640,
0x7200, 0xb2c1, 0xb381, 0x7340, 0xb101, 0x71c0, 0x7080, 0xb041,
0x5000, 0x90c1, 0x9181, 0x5140, 0x9301, 0x53c0, 0x5280, 0x9241,
0x9601, 0x56c0, 0x5780, 0x9741, 0x5500, 0x95c1, 0x9481, 0x5440,
0x9c01, 0x5cc0, 0x5d80, 0x9d41, 0x5f00, 0x9fc1, 0x9e81, 0x5e40,
0x5a00, 0x9ac1, 0x9b81, 0x5b40, 0x9901, 0x59c0, 0x5880, 0x9841,
0x8801, 0x48c0, 0x4980, 0x8941, 0x4b00, 0x8bc1, 0x8a81, 0x4a40,
0x4e00, 0x8ec1, 0x8f81, 0x4f40, 0x8d01, 0x4dc0, 0x4c80, 0x8c41,
0x4400, 0x84c1, 0x8581, 0x4540, 0x8701, 0x47c0, 0x4680, 0x8641,
0x8201, 0x42c0, 0x4380, 0x8341, 0x4100, 0x81c1, 0x8081, 0x4040
};

/*
        procdure: calc_crc

        author: Marvin Sojka

        created: 2/29/87

        history:

        purpose: calculate crc-16 values for a string of data of giv
en length.

        paramters:
                        buff -- pointer to buffer array.
                        len  -- length to calculate the crc-16 value
 on.

        returns:
                        crc-16 value.
*/
WORD calc_crc(buff,len)
BYTE *buff;
WORD len;
{
        WORD index,crc;
        crc = 0;
        while (len)
        {
                index = ((crc ^ (*buff & 0xff)) & 0x00ff);
                crc = ( (crc >> 8) & 0x00ff) ^ crc_table[index];
                len--;
                buff++;
        }
        return (crc);

}


WORD calc_rx_crc(crc,new)
WORD crc;
BYTE new;
{
        WORD index;
```

```
        index = ((crc ^ (new & 0xff)) & 0x00ff);
        return(((crc >> 8) & 0x00ff) ^ crc_table[index]);
}
```

```
/*****************************************************************************
****

Copyright (C)1991, Norand Corporation.  All Rights Reserved.


Revision History:

** $Log:   N:/3250/rtc/src/main.c_v  $
 *
 *    Rev 1.6   27 Feb 1992 16:06:08   SOJKA
 * Dual Host Support Release and RS-422 Bug
 *
 *    Rev 1.5   18 Feb 1992 10:47:46   SOJKA
 * UIF cosmetic fixes V0.23
 *
 *    Rev 1.4   31 Dec 1991 11:22:58   sojka
 * Complied with Library V0.21.
 *
 *    Rev 1.3   31 Dec 1991 10:40:48   sojka
 * Complied with Library V0.21.
 *
 *    Rev 1.2   20 Nov 1991 18:37:42   sojka
 * Added Copyright Information and PVCS Log keyword.

*****************************************************************************
****/



#include "main.h"
#include <68302.h>
#include <lcd.h>

/*
    main
    This is the main startup routine for rtc master.
        It initialize the data for default mode.
        It checkes input paramters for usage.
    It also creates the inactivity timer task.  Last thing it does
    is set down the priority of the task and call rad_main,
    the main processing loop of rtc.

*/
rad_cfg_ptr rad_cfgx;

void main(x)
rad_cfg_ptr x;
{

        rad_cfgx = x;

/*      com_start(0x10);
*/      /* initial startup queues */
        queues_start();
```

```
        /* initialize buffer start */
        buff_start();
        merr_start();

        /* init LCD access */
        lcd_startup();

        /* init mailbox */
        mailbox_init();
        /* intialize RCB packages */
    rcb_init();
        /* initialize some rtc variables */
        rtc_init();

        /* initialize timers */
    rad_timer_init();
        /* start scc */
        scc_setup();

        /* initialize statistics */
        stats_init();

        /* bump priority down */
    set_priority(RTC_PRIORITY);

        callsign_set(&rad_cfgx->fcc_callsign[0]);

    /* start the real processing */
    rad_main();

}
```

```
/*************************************************************************
****

Copyright (C)1991, Norand Corporation.  All Rights Reserved.


Revision History:

** $Log:   N:/3250/rtc/src/rad.c_v  $
 *
 *    Rev 1.6   27 Feb 1992 16:06:04   SOJKA
 * Dual Host Support Release and RS-422 Bug
 *
 *    Rev 1.5   18 Feb 1992 10:47:44   SOJKA
 * UIF cosmetic fixes V0.23
 *
 *    Rev 1.4   31 Dec 1991 11:22:56   sojka
 * Complied with Library V0.21.
 *
 *    Rev 1.3   31 Dec 1991 10:40:46   sojka
 * Complied with Library V0.21.
 *
 *    Rev 1.2   20 Nov 1991 18:37:40   sojka
 * Added Copyright Information and PVCS Log keyword.

*************************************************************************
****/



#include "main.h"

/*
        rad.c
        author Marvin Sojka

        purpose:
                Declarations for global identies used in the RTC pro
tocol
*/

void (*rad_timer_routine)();
rcb rcbs[MAX_TERMS];

WORD up_q[MAX_TERMS];
WORD up_q_count;
WORD up_flag;
WORD rtc_slots;

rcb_ptr  crcb;
buff_ptr cbuff;

WORD     squelch[2];                          /* index by rx_port
*/
WORD     squelch_check[2];
```

```
buff_ptr rtc_rx_buff[2];
BYTE     *rtc_rxptr[2];
BYTE     *rtc_rxptr1[2];
WORD     rtc_rx_length[2];
WORD     rtc_rxx[2];
WORD     rad_state[2];
WORD     rtc_rx_crc[2];

BYTE rad_tx_buffer[300];
WORD rad_tx_length;
BYTE rad_rx_buffer[300];
BYTE rad_rx_buffer1[300];


WORD rad_timer;
WORD rad_inact_timeout;

WORD rad_callsign_timer;

WORD port;
WORD base;


/* stats */
LONG    term_out_blocks[MAX_TERMS];
LONG    term_in_blocks[MAX_TERMS];
LONG    term_out_timeouts[MAX_TERMS];
LONG    term_in_timeouts[MAX_TERMS];

LONG    glob_out_blocks;
LONG    glob_in_blocks;
LONG    glob_out_timeouts;
LONG    glob_in_timeouts;
LONG    glob_interference;
LONG    glob_collisions;

LONG    channel_out_blocks[4];
LONG    channel_in_blocks[4];
LONG    channel_out_timeouts[4];
LONG    channel_in_timeouts[4];
LONG    channel_interference[4];
LONG    channel_collisions[4];
```

```
/*************************************************************************
****

Copyright (C)1991, Norand Corporation.  All Rights Reserved.

Revision History:

** $Log:   N:/3250/rtc/src/rad_in.c_v  $
 *
 *    Rev 1.6   27 Feb 1992 16:05:26   SOJKA
 * Dual Host Support Release and RS-422 Bug
 *
 *    Rev 1.5   18 Feb 1992 10:47:06   SOJKA
 * UIF cosmetic fixes V0.23
 *
 *    Rev 1.4   31 Dec 1991 11:22:16   sojka
 * Complied with Library V0.21.
 *
 *    Rev 1.3   31 Dec 1991 10:40:06   sojka
 * Complied with Library V0.21.
 *
 *    Rev 1.2   20 Nov 1991 18:37:00   sojka
 * Added Copyright Information and PVCS Log keyword.

*************************************************************************
****/



#include "main.h"


/*
    module:   rtc_in.c

    procedure: rtc_in

    author: Marvin Sojka

    created: 2/29/87

    called by:
        rtc_main.

    calls:
        rcb_find
        rcb_free
        rcb_alloc

    history:

    purpose: this routine is handles the data from the host to proce
ssed.
```

```
*/

void rad_in()
{
    buff_ptr qbuff;
    rcb_ptr  qrcb;
    WORD i;

        /* check if a buffer is available
           if not return
        */
    while ((qbuff = check_main())!=NULL)
    {
        switch (qbuff->type)
        {

                        /* RF_TEST, RF_VERSION, and RF_DATA are queu
e to the
                                terminals rcb if one is active
                        */
                        case RF_TEST:
            case RF_DATA:
                        case RF_VERSION:
                        case RF_ECHO:
                                rf_data_in(qbuff);
                break;

                        /*
                                RF_REBOOT resets the rcb
                        */
            case RF_REBOOT:
                                rf_reboot_in(qbuff);
                                break;

                        /*
                                RF_ENABLE will allocate a rcb for a
given address
                                if one doesn't exist or resets the r
cb if one already
                                does. A RF_ENABLE for the general us
e (address = -1)
 data buffer
                                is processed by each position in the
                                corresponds to the terminal address
and if that
                                position is positive then it is enab
led otherwise
                                it is disabled.
                        */
            case RF_ENABLE:
                                rf_enable_in(qbuff);
                break;

            /*
                                RF_DISABLE will take a rcb off the a
```

```
ctive list
                        */
                case RF_DISABLE:
                    if ((qrcb = rcb_find(qbuff->address)) != NULL)
                    {
                          rcb_free(qrcb);
                    }
                    buff_free(qbuff);
                    break;

                        /*
                                RF_MSTAT will find the state of a rc
b txq
                                        1 -- not active or allocated
                                        2 -- Active
                                        3 -- Message
                                        4 -- Inactive with Message
                        */
                case RF_MSTAT:
                                rf_mstat_in(qbuff);
                    break;

                        /*
                                RF_HALT will stop the RTC task until
 a RF_GO
                                command is received.
                        */
                        case RF_HALT:
                                buff_free(qbuff);
                                rf_halt_in();
                                break;

                        /*
                                RF_CSS passes in the data portion of
 the buffer
                                the string to use for FCC Callsign p
rocessing
                        */
                        case RF_CCS:
                                qbuff->data[qbuff->length] = '\0';
                                callsign_set(&qbuff->data[0]);
                                buff_free(qbuff);
                                break;

                        /*
                                RF_PORT enables BASE, AUX or DUAL_PO
RT or MBA support
                        */
                        case RF_PORT:
                                rad_cfgx->base = qbuff->data[0] & 0x
FF;
                                if (rad_cfgx->base == MBA)
                                {
                                        rad_cfgx->mba[0] = qbuff->da
ta[1];
                                        rad_cfgx->mba[1] = qbuff->da
```

```
ta[2];
                                        rad_cfgx->mba[2] = qbuff->da
ta[3];
                                        rad_cfgx->mba[3] = qbuff->da
ta[4];
                                }
                                scc_setup();
                                buff_free(qbuff);
                                break;

                        /*
                                RF_SPEED chnages RF baud rate for su
pport
                        */
                        case RF_SPEED:
                                rad_cfgx->speed[0] = qbuff->data[0]
& 0xFF;
                                rad_cfgx->speed[1] = qbuff->data[1]
& 0xFF;
                                rad_cfgx->speed[2] = qbuff->data[2]
& 0xFF;
                                rad_cfgx->speed[3] = qbuff->data[3]
& 0xFF;
                                buff_free(qbuff);
                                break;

                        /*
                                RF_STAT will return the terminal or
mux global statisitics
                        */
                        case RF_STAT:
                                rf_stat_in(qbuff);
                    break;

                        /*
                                RF_STAT_RESET will reset terminal or
 mux global statisitics
                        */
                        case RF_STAT_RESET:
                                rf_stat_reset_in(qbuff);
                                break;


                        /*
                                RF_BASE_POS set remote base parametr
s
                        */
                        case RF_BASE_POS:
                                if (qbuff->data[0] & 0x01)
                                        rad_cfgx->remote_base[0] = R
EMOTE_ON;
                                else
                                        rad_cfgx->remote_base[0] = R
EMOTE_OFF;
                                if (qbuff->data[0] & 0x02)
                                        rad_cfgx->remote_base[1] = R
```

```
EMOTE_ON;
                                        else
                                                rad_cfgx->remote_base[1] = R
EMOTE_OFF;
                                        if (qbuff->data[0] & 0x04)
                                                rad_cfgx->remote_base[2] = R
EMOTE_ON;
                                        else
                                                rad_cfgx->remote_base[2] = R
EMOTE_OFF;
                                        if (qbuff->data[0] & 0x08)
                                                rad_cfgx->remote_base[3] = R
EMOTE_ON;
                                        else
                                                rad_cfgx->remote_base[3] = R
EMOTE_OFF;
                                        rad_cfgx->remote_offset[0] = qbuff->
data[1];
                                        rad_cfgx->remote_offset[1] = qbuff->
data[2];
                                        rad_cfgx->remote_offset[2] = qbuff->
data[3];
                                        rad_cfgx->remote_offset[3] = qbuff->
data[4];
                                        buff_free(qbuff);
                                        break;

                                /*
                                        RF_TYPE will return the terminal typ
e of a
                                        terminal that was detedted at startu
p.
                                */
                        case RF_TYPE:
                                        rf_type_in(qbuff);
                                        break;

                                default:
                                buff_free(qbuff);
                                break;
                        }
                }
        }

void rf_halt_in()
{
        buff_ptr qbuff;
        rcb_ptr  qrcb;
        WORD i;

                /* loop until go */
                for (;;)
                {
                        lcd_update();
                        lcd_sys_status(RAD_HOST);
                        /* check if a buffer is available
```

```
                */
                while ((qbuff = wait_main(10))!=NULL)
                {
                        switch (qbuff->type)
                        {

                        /* RF_TEST, RF_VERSION, and RF_DATA are queu
e to the
                                terminals rcb if one is active
                        */
                        case RF_TEST:
                        case RF_DATA:
                        case RF_VERSION:
                        case RF_ECHO:
                                rf_data_in(qbuff);
                                break;

                        /*
                                RF_REBOOT resets the rcb
                        */
                        case RF_REBOOT:
                                rf_reboot_in(qbuff);
                                break;

                        /*
                                RF_ENABLE will allocate a rcb for a
given address
                                if one doesn't exist or resets the r
cb if one already
                                does. A RF_ENABLE for the general us
e (address = -1)
                                is processed by each position in the
 data buffer
                                corresponds to the terminal address
and if that
                                position is positive then it is enab
led otherwise
                                it is disabled.
                        */
                case RF_ENABLE:
                                rf_enable_in(qbuff);
                    break;

                /*
                                RF_DISABLE will take a rcb off the a
ctive list
                        */
                case RF_DISABLE:
                    if ((qrcb = rcb_find(qbuff->address)) != NULL)
                    {
                          rcb_free(qrcb);
                    }
                    buff_free(qbuff);
                    break;

                        /*
```

```
                                RF_MSTAT will find the state of a rc
b txq
                                        1 -- not active or allocated
                                        2 -- Active
                                        3 -- Message
                                        4 -- Inactive with Message
                        */
                case RF_MSTAT:
                                rf_mstat_in(qbuff);
                    break;

                        /*
                                RF_HALT will stop the RTC task until
 a RF_GO
                                command is received.
                        */
                        case RF_HALT:
                                buff_free(qbuff);
                                break;

                        case RF_GO:
                                buff_free(qbuff);
                                return;

                        /*
                                RF_CSS passes in the data portion of
 the buffer
                                the string to use for FCC Callsign p
rocessing
                        */
                        case RF_CCS:
                                qbuff->data[qbuff->length] = '\0';
                                callsign_set(&qbuff->data[0]);
                                buff_free(qbuff);
                                break;

                        /*
                                RF_PORT enables BASE, AUX or DUAL_PO
RT or MBA support
                        */
                        case RF_PORT:
                                rad_cfgx->base = qbuff->data[0] & 0x
FF;
                                if (rad_cfgx->base == MBA)
                                {
                                        rad_cfgx->mba[0] = qbuff->da
ta[1];
                                        rad_cfgx->mba[1] = qbuff->da
ta[2];
                                        rad_cfgx->mba[2] = qbuff->da
ta[3];
                                        rad_cfgx->mba[3] = qbuff->da
ta[4];
                                }
                                scc_setup();
                                buff_free(qbuff);
```

```
                        break;

                /*
                        RF_SPEED chnages RF baud rate for su
pport
                */
                case RF_SPEED:
                        rad_cfgx->speed[0] = qbuff->data[0]
& 0xFF;
                        rad_cfgx->speed[1] = qbuff->data[1]
& 0xFF;
                        rad_cfgx->speed[2] = qbuff->data[2]
& 0xFF;
                        rad_cfgx->speed[3] = qbuff->data[3]
& 0xFF;
                        buff_free(qbuff);
                        break;

                /*
                        RF_STAT will return the terminal or
mux global statisitics
                */
                case RF_STAT:
                        rf_stat_in(qbuff);
                break;

                /*
                        RF_STAT_RESET will reset terminal or
 mux global statisitics
                */
                case RF_STAT_RESET:
                        rf_stat_reset_in(qbuff);
                        break;


                /*
                        RF_BASE_POS set remote base parametr
s
                */
                case RF_BASE_POS:
                        if (qbuff->data[0] & 0x01)
                                rad_cfgx->remote_base[0] = R
EMOTE_ON;
                        else
                                rad_cfgx->remote_base[0] = R
EMOTE_OFF;
                        if (qbuff->data[0] & 0x02)
                                rad_cfgx->remote_base[1] = R
EMOTE_ON;
                        else
                                rad_cfgx->remote_base[1] = R
EMOTE_OFF;
                        if (qbuff->data[0] & 0x04)
                                rad_cfgx->remote_base[2] = R
EMOTE_ON;
                        else
```

```
                                        rad_cfgx->remote_base[2] = R
EMOTE_OFF;
                                if (qbuff->data[0] & 0x08)
                                        rad_cfgx->remote_base[3] = R
EMOTE_ON;
                                else
                                        rad_cfgx->remote_base[3] = R
EMOTE_OFF;
                                rad_cfgx->remote_offset[0] = qbuff->
data[1];
                                rad_cfgx->remote_offset[1] = qbuff->
data[2];
                                rad_cfgx->remote_offset[2] = qbuff->
data[3];
                                rad_cfgx->remote_offset[3] = qbuff->
data[4];
                                buff_free(qbuff);
                                break;

                        /*
                                RF_TYPE will return the terminal typ
e of a
                                terminal that was detedted at startu
p.
                        */
            case RF_TYPE:
                                rf_type_in(qbuff);
                                break;

                        default:
                buff_free(qbuff);
                break;
                        }
                }
    }
}

/*
        rf_data_in
        checks to see if a rcb is active for this address a
        and queue data to it if possilble
*/
void rf_data_in(qbuff)
buff_ptr qbuff;
{
        buff_ptr temp;
    rcb_ptr  qrcb;
        /* Check to see if a rcb is active for this address */
    if ((qrcb = rcb_find(qbuff->address)) != NULL)
    {
                /* chain if NULL, easy just queue it */
        if (qrcb->txq == NULL)
            qrcb->txq = qbuff;
        else
        {
                        /* if used for SNA then
```

```
                                queue can have multiple chains
                                otherwise release the current chain
and
                                send the next one
                        */
                        if (rad_cfgx->sna)
                        {
                                if (qrcb->long_timer == 0)
                                {
                                        buff_free(qrcb->txq);
                                        qrcb->txq = qbuff;
                                }
                                else
                                {
                                        temp = qrcb->txq;
                                        while (temp->next != NULL)
                                                temp = temp->next;
                                        temp->next = qbuff;
                                }
                        }
                        else
                        {
                                buff_free(qrcb->txq);
                                qrcb->txq = qbuff;
                        }
                }
                        qrcb->long_timer = 60;
                        qrcb->inact_timer = rad_inact_timeout;
        }
        else
                {
                        if (qbuff->type == RF_ECHO)
                        {
                               qrcb = rcb_alloc(qbuff->address);
                               qrcb->long_timer = 60;
                               qrcb->inact_timer = rad_inact_timeout;
                               qrcb->txq=qbuff;
                        }
                        else
                                buff_free(qbuff);
                }
}

/*
        rf_reboot_in
        checks for mux address. if it is then reboot all terminals
        otherwise reboot only the single terminal
*/
void rf_reboot_in(qbuff)
buff_ptr qbuff;
{
    rcb_ptr  qrcb;
        WORD i;


        if (qbuff->address != RF_MUX)
```

```
        {
                /* check for already allocated terminal.  if not ena
ble one */
                if ((qrcb = rcb_find(qbuff->address)) != NULL)
                        rtc_reset(qrcb);
                else
                        qrcb = rcb_alloc(qbuff->address);
        }
        else
        {
                /* reset all active terminals */
                for (i=0;i<MAX_TERMS;i++)
                {
                        qrcb = &rcbs[i];
                        if (qrcb->free == RCB_INUSE)
                                rtc_reset(qrcb);
        }
    }
    buff_free(qbuff);
}


/*
        rf_enable_in
        will allocate a rcb for a given address
        if one doesn't exist or resets the rcb if one already
        does. A RF_ENABLE for the general use (address = -1)
        is processed by each position in the data buffer
        corresponds to the terminal address and if that
        position is positive then it is enabled otherwise
        it is disabled.
*/
void rf_enable_in(qbuff)
buff_ptr qbuff;
{
    rcb_ptr  qrcb;
        WORD i;

        if (qbuff->address != RF_MUX)
        {
                /* check for a valid address */
                if ((qrcb = rcb_find(qbuff->address)) == NULL)
                {
                        if (qbuff->data[0] == 1)
                                qrcb = rcb_alloc(qbuff->address);
                }
                else
                {
                        if (qbuff->data[0] == 1)
                        {
                                if (!(rad_cfgx->sna))
                                        rtc_reset(qrcb);
                        }
                        else
                                rcb_free(qrcb);
                }
```

```
        }
        else
        {
                /* check for length of buffer */
                for (i=0;i<qbuff->length;i++)
                {
                        /* if a rcb is not found with the address */
                        if ((qrcb = rcb_find(i)) == NULL)
                        {
                                /* check for an enable command */
                                if (qbuff->data[i] == 1)
                                {
                                        qrcb = rcb_alloc(i);
                                }
                        }
                        else
                        {
                                /* check for a disable command */
                                if (qbuff->data[i] == 0)
                                        rcb_free(qrcb);
                        }
                }
        }
        buff_free(qbuff);
}

/*
        rf_mstat_in
        will find the state of the rcb in relation to the rcb txq as
 follows
                1 -- not active or allocated (MSTAT_NOTACTIVE)
                2 -- Active                                        (MS
TAT_ACTIVE)
                3 -- Message                                  (MSTAT_MESS
AGE)
                4 -- Inactive with Message          (MSTAT_INACTIVE_MES
SAGE)
                5 -- inactive                                 (MSTAT_INAC
TIVE)

        The global use flag (address = -1) will return the value for
        all terminals.
*/
void rf_mstat_in(qbuff)
buff_ptr qbuff;
{
    rcb_ptr  qrcb;
        WORD j;
        BYTE i;


        if (qbuff->address != RF_MUX)
        {
                i = MSTAT_NOT_ACTIVE;
                /* check for a active terminal */
                if ((qrcb = rcb_find(qbuff->address)) != NULL)
```

```
                {
                        /* check for message in queue */
                        if (qrcb->txq != NULL)
                        {
                                /* check for inactivity */
                                if (qrcb->inact_timer == 0)
                                        i = MSTAT_INACTIVE_MESSAGE;
                                else
                                        i = MSTAT_MESSAGE;
                        }
                        else if (qrcb->inact_timer == 0)
                                i = MSTAT_INACTIVE;
                        else
                                i = MSTAT_ACTIVE;
                }
                /* send back to upper layers */
                qbuff->chain_status = OIC;
                qbuff->length = 1;
                qbuff->data[0] = i;
                /* if from is not null then assume this is
                   the address for for a post of the buffer */
                if (qbuff->from != NULL)
                        post_main(qbuff,qbuff->from);
                else
                        send_main(qbuff);
        }
        else
        {
                        qbuff->data[j] = MSTAT_NOT_ACTIVE;

                /* check all terminals for activity */
                for (j=0;j<MAX_TERMS;j++)
                {
                        qrcb = &rcbs[j];
                        /* check terminal if it is allocated */
                        if (qrcb->free == RCB_INUSE)
                        {
                                /* check for buffer */
                                if (qrcb->txq != NULL)
                                {
                                        /* check for inactivity */
                                        if (qrcb->inact_timer == 0)
                                                i = MSTAT_INACTIVE_M
ESSAGE;
                                        else
                                                i = MSTAT_MESSAGE;
                                }
                                else if (qrcb->inact_timer == 0)
                                        i = MSTAT_INACTIVE;
                                else
                                        i = MSTAT_ACTIVE;
                        }
                        else
                                i = MSTAT_NOT_ACTIVE;
                        qbuff->data[j] = i;
```

```
                }
                qbuff->chain_status = OIC;
                qbuff->length = MAX_TERMS;
                /* if from is not null then assume this is
                   the address for for a post of the buffer */
                if (qbuff->from != NULL)
                        post_main(qbuff,qbuff->from);
                else
                        send_main(qbuff);
        }
}

/*
        rf_stat_in
        will return the terminal or mux global statisitics
*/
void rf_stat_in(qbuff)
buff_ptr qbuff;
{
        WORD i,k;
        if (qbuff->address == RF_MUX)
        {
                convert_long_to_bytes(&qbuff->data[GLOB_OUT_BLOCKS],
                                                        glob_out_b
locks);
                convert_long_to_bytes(&qbuff->data[GLOB_IN_BLOCKS],
                                                        glob_in_bl
ocks);
                convert_long_to_bytes(&qbuff->data[GLOB_OUT_TIMEOUTS
],
                                                        glob_out_t
imeouts);
                convert_long_to_bytes(&qbuff->data[GLOB_IN_TIMEOUTS]
,
                                                        glob_in_ti
meouts);
                convert_long_to_bytes(&qbuff->data[GLOB_INTERFERENCE
],
                                                        glob_inter
ference);
                convert_long_to_bytes(&qbuff->data[GLOB_COLLISIONS],
                                                        glob_colli
sions);
                k = STAT_SIZE;
                for (i=0;i<4;i++)
                {
                        convert_long_to_bytes(&qbuff->data[k+GLOB_OU
T_BLOCKS],
                                                        channel_ou
t_blocks[i]);
                        convert_long_to_bytes(&qbuff->data[k+GLOB_IN
_BLOCKS],
                                                        channel_in
_blocks[i]);
                        convert_long_to_bytes(&qbuff->data[k+GLOB_OU
T_TIMEOUTS],
```

```
                                                                channel_ou
t_timeouts[i]);
                        convert_long_to_bytes(&qbuff->data[k+GLOB_IN
_TIMEOUTS],
                                                                channel_in
_timeouts[i]);
                        convert_long_to_bytes(&qbuff->data[k+GLOB_IN
TERFERENCE],
                                                                channel_in
terference[i]);
                        convert_long_to_bytes(&qbuff->data[k+GLOB_CO
LLISIONS],
                                                                channel_co
llisions[i]);
                        k += STAT_SIZE;
                }
        }
        else
        {
                convert_long_to_bytes(&qbuff->data[TERM_OUT_BLOCKS],
                                                                term_out_b
locks[qbuff->address]);
                convert_long_to_bytes(&qbuff->data[TERM_IN_BLOCKS],
                                                                term_in_bl
ocks[qbuff->address]);
                convert_long_to_bytes(&qbuff->data[TERM_OUT_TIMEOUTS
],
                                                                term_out_t
imeouts[qbuff->address]);
                convert_long_to_bytes(&qbuff->data[TERM_IN_TIMEOUTS]
,
                                                                term_in_ti
meouts[qbuff->address]);
        }
        post_main(qbuff,qbuff->from);
}


/*
        rf_stat_reset_in
        will reset terminal or mux global statisitics
*/
void rf_stat_reset_in(qbuff)
buff_ptr qbuff;
{
        if (qbuff->address == RF_MUX)
                stats_init();
        else
                clear_term_stats(qbuff->address);
        buff_free(qbuff);
}

/*
        rf_type_in
        returns the type value that the mux detected and saved in th
e reset
```

```
        message or powerup message from a terminal.  the global use
flag(-1)
        enables the return of all terminal type values.
*/
void rf_type_in(qbuff)
buff_ptr qbuff;
{
    rcb_ptr  qrcb;
        BYTE i;
        WORD j;

        if (qbuff->address != RF_MUX)
        {
                i = 0;
                /* check for a active terminal */
                if ((qrcb = rcb_find(qbuff->address)) != NULL)
                {
/*                      if ((protocol_level != 0) &&
                            (qbuff->address < NATIVE_LIMIT))
                        {
                                if (qrcb->fastrad)
                                        i = 2;
                                else
                                        i = 1;
                        }
                        else
*/                              i = (BYTE)qrcb->type;
                }
                        qbuff->data[0] = i;
                        /* send back to upper layers */
                        qbuff->chain_status = OIC;
                        qbuff->length = 1;
                        qbuff->data[0] = i;
                        /* if from is not null then assume this is
                           the address for for a post of the buffer
*/
                        if (qbuff->from != NULL)
                                post_main(qbuff,qbuff->from);
                        else
                                send_main(qbuff);
        }
        else
        {
                /* set all terminals to inactive */
               for (j=0;j<MAX_TERMS;j++)
                        qbuff->data[j] = 0;

                /* check all terminals for activity */
                for (j=0;j<MAX_TERMS;j++)
                {
                        qrcb = &rcbs[j];
                        /* check terminal if it is allocated */
                        if (qrcb->free == 1)
                        {
/*                              if ((protocol_level != 0) &&
                                        (qbuff->address < NATIVE_LIM
```

```
IT))
                                {
                                        if (qrcb->fastrad)
                                                i = 2;
                                        else
                                                i = 1;
                                }
                                else
                                        i = (BYTE)qrcb->type;
*/
                                qbuff->data[j] = i;
                        }
                }
                qbuff->chain_status = OIC;
                qbuff->length = MAX_TERMS;
                /* if from is not null then assume this is
                   the address for for a post of the buffer */
                if (qbuff->from != NULL)
                        post_main(qbuff,qbuff->from);
                else
                        send_main(qbuff);
        }
}

convert_long_to_bytes(x,y)
BYTE *x;
LONG y;
{
        *x++ = (y>>24) & 0xFF;
        *x++ = (y>>16) & 0xFF;
        *x++ = (y>>8) & 0xFF;
        *x++ = y & 0xFF;
}
```

```
/*************************************************************************
****

Copyright (C)1991, Norand Corporation.  All Rights Reserved.


Revision History:

** $Log:   N:/3250/rtc/src/rad_inf.c_v  $
 *
 *    Rev 1.6   27 Feb 1992 16:05:22   SOJKA
 * Dual Host Support Release and RS-422 Bug
 *
 *    Rev 1.5   18 Feb 1992 10:47:02   SOJKA
 * UIF cosmetic fixes V0.23
 *
 *    Rev 1.4   31 Dec 1991 11:22:12   sojka
 * Complied with Library V0.21.
 *
 *    Rev 1.3   31 Dec 1991 10:40:04   sojka
 * Complied with Library V0.21.
 *
 *    Rev 1.2   20 Nov 1991 18:36:58   sojka
 * Added Copyright Information and PVCS Log keyword.

*************************************************************************
****/



#include <amx581sd.h>
#include "main.h"

extern rad_cfg_ptr rad_cfgx;
LONG sys_mailbox;
AMXID mailboxid;

/*
        send_main
        communicates a buffer from rtc to main task
*/
void send_main(x)
buff_ptr x;
{
        queue_put(rad_cfgx->rad_out_q,(queue_elem_ptr)x);
}

/*
        wait_in
        wait for a packet from main task
*/
buff_ptr wait_main(timecut)
LONG timeout;
{
        return((buff_ptr)queue_wait(rad_cfgx->rad_in_q,timeout));
```

```
}

/*
        check_in
        check for a packet from main task
*/
buff_ptr check_main()
{
        buff_ptr temp;
        temp =(buff_ptr)queue_get(rad_cfgx->rad_in_q);

        return(temp);
}

void post_main(x,que)
buff_ptr x;
queue_ptr que;
{
        queue_put(que,(queue_elem_ptr)x);
}

/*
        post mailbox
*/
post_mailbox(x)
LONG x;
{
        sys_mailbox = x;
        ajwake(mailboxid);
}
/*
        pend mailbox
*/
LONG pend_mailbox(x)
LONG x;
{
        sys_mailbox = 0;
        ajwapr();
        if (x==0)
                ajwait();
        else
                ajwatm(x);
        return(sys_mailbox);
}
void mailbox_init()
{
        sys_mailbox = 0;
        mailboxid = ajtkid();
}
```

```
/*****************************************************************************
****

Copyright (C)1991, Norand Corporation.  All Rights Reserved.


Revision History:

** $Log:   N:/3250/rtc/src/rad_lcd.c_v  $
 *
 *    Rev 1.6   27 Feb 1992 16:05:44   SOJKA
 * Dual Host Support Release and RS-422 Bug
 *
 *    Rev 1.5   18 Feb 1992 10:47:22   SOJKA
 * UIF cosmetic fixes V0.23
 *
 *    Rev 1.4   31 Dec 1991 11:22:34   sojka
 * Complied with Library V0.21.
 *
 *    Rev 1.3   31 Dec 1991 10:40:24   sojka
 * Complied with Library V0.21.
 *
 *    Rev 1.2   20 Nov 1991 18:37:20   sojka
 * Added Copyright Information and PVCS Log keyword.

*****************************************************************************
****/



#include "main.h"
#include <lcd.h>


BYTE LCD_tx_status[] = "SPDTVR*C";
BYTE LCD_rx_status[] = "GTC";
BYTE LCD_config[]    = "RDMSX";

/*
        routines to support the LCD display for RTC
*/

WORD lcd_lock;

/*
        lcd_startup
        called at RTC startup to initialize LCD access
*/

void lcd_startup()
{
        /* get connection */
        LCD_start();
        if (lcd_lock = LCD_lock(CHECK_LOCK))
        {
```

```
                LCD_clear();
                LCD_cursor_off();
                LCD_printf("ADR T R HB P SPD PR STAT");
        }
}

void lcd_update()
{
        WORD i;

        /* release lock */
        LCD_unlock();

        /* check for access */
        if ((i = LCD_lock(CHECK_LOCK)) && (!lcd_lock))
        {
                LCD_clear();
                LCD_cursor_off();
                LCD_printf("ADR T R HB P SPD PR STAT");
        }
        lcd_lock = i;
}

/*
        lcd_addr
        display addr on lcd display
*/
void lcd_addr(addr)
WORD addr;
{
        if (lcd_lock)
        {
                LCD_set_cursor(1,0);
                LCD_printf("%03d",addr);
        }
}

/*
        lcd_tx_status
        display tx type
*/
void lcd_tx_status(status)
WORD status;
{
        if (lcd_lock)
        {
                LCD_set_cursor(1,4);
                LCD_putc(LCD_tx_status[status]);
        }
}

/*
        lcd_rx_status
        display rx type
*/
void lcd_rx_status(status)
```

```
WORD status;.
{
        if (lcd_lock)
        {
                LCD_set_cursor(1,6);
                LCD_putc(LCD_rx_status[status]);
        }
}

/*
        lcd_heartbeats
        display # heartbeats
*/
void lcd_heartbeats(heartbeats)
WORD heartbeats;
{
        if (lcd_lock)
        {
                LCD_set_cursor(1,8);
                LCD_printf("%2d",heartbeats);
        }
}

/*
        lcd_port
        display port accessing number
*/
void lcd_port(port)
WORD port;
{
        if (lcd_lock)
        {
                LCD_set_cursor(1,11);
                LCD_putc(port + '0');
        }
}

/*
        lcd_speed
        display rf speed
*/
void lcd_speed(speed)
WORD speed;
{
        if (lcd_lock)
        {
                LCD_set_cursor(1,13);
                if (speed == LCD_4800)
                        LCD_puts((BYTE *)"4.8");
                else
                        LCD_puts((BYTE *)"9.6");
        }
}

/*
        lcd_config
```

```
        display config information
*/
void lcd_config(config)
WORD config;
{
        if (lcd_lock)
        {
                LCD_set_cursor(1,17);
                LCD_putc(LCD_config[config]);

        }
}


/*
        lcd_sys_status
        indicates good, bad radio or bad host
*/
void lcd_sys_status(status)
WORD status;
{

        if (lcd_lock)
        {
                LCD_set_cursor(1,20);
                switch (status)
                {

                        case RAD_GOOD:
                                LCD_puts((BYTE *)"GOOD");
                                break;

                        case RAD_RADIO:
                                LCD_puts((BYTE *)"BASE");
                                break;

                        case RAD_HOST:
                                LCD_puts((BYTE *)"HOST");
                                break;
                }
        }
}
```

```
/*****************************************************************************
****

Copyright (C)1991, Norand Corporation.  All Rights Reserved.

Revision History:

** $Log:   N:/3250/rtc/src/rad_main.c_v  $
 *
 *    Rev 1.6   27 Feb 1992 16:05:36   SOJKA
 * Dual Host Support Release and RS-422 Bug
 *
 *    Rev 1.5   18 Feb 1992 10:47:14   SOJKA
 * UIF cosmetic fixes V0.23
 *
 *    Rev 1.4   31 Dec 1991 11:22:26   sojka
 * Complied with Library V0.21.
 *
 *    Rev 1.3   31 Dec 1991 10:40:18   sojka
 * Complied with Library V0.21.
 *
 *    Rev 1.2   20 Nov 1991 18:37:12   sojka
 * Added Copyright Information and PVCS Log keyword.

*****************************************************************************
****/



#include "main.h"

/*
   rad_main
   procesing loop for rtc
*/

void rad_main()
{

    for (;;)
    {
                lcd_update();
                callsign_check();
        rad_in();
                rtc_main();
        }
}
```

```
/*****************************************************************************
****

Copyright (C)1991, Norand Corporation.  All Rights Reserved.


Revision History:

** $Log:   N:/3250/rtc/src/rad_rcb.c_v  $
 *
 *    Rev 1.6   27 Feb 1992 16:05:48   SOJKA
 * Dual Host Support Release and RS-422 Bug
 *
 *    Rev 1.5   18 Feb 1992 10:47:26   SOJKA
 * UIF cosmetic fixes V0.23
 *
 *    Rev 1.4   31 Dec 1991 11:22:40   sojka
 * Complied with Library V0.21.
 *
 *    Rev 1.3   31 Dec 1991 10:40:30   sojka
 * Complied with Library V0.21.
 *
 *    Rev 1.2   20 Nov 1991 18:37:22   sojka
 * Added Copyright Information and PVCS Log keyword.

*****************************************************************************
****/



#include "main.h"


/*
    module:   rad_rcb.c

    author: Marvin Sojka

    created: 2/29/87

    history:

    purpose: handles initialize, allocating, freeing and finding rcb
's.
             rcb_alloc
             rcb_free
             rcb_find
             rcb_init
*/

/*
    procdure: rtc_alloc

    author: Marvin Sojka
```

```
    created: 2/29/87

    called by:
        rtc_main
        rtc_in
        rtc_reset

    calls:
        none

    history:

    purpose: This is allocs a rcb with the address passed.  If no rc
b is
             available then a NULL is returned.

*/
rcb_ptr rcb_alloc(address)
WORD address;
{
        WORD i;
    rcb_ptr qrcb;
        BYTE *k;

        if (address >= MAX_TERMS)
                return NULL;
        else if (rcbs[address].free == 0)
    {
        qrcb = &rcbs[address];
        qrcb->free = 1;
        qrcb->address = address;
        qrcb->state = RESET_STATE;
        qrcb->vr = 0;
        qrcb->vs = 0;
        qrcb->txq = NULL;
        qrcb->rxq = NULL;
        qrcb->rx_length = 0;
                qrcb->retrys = 0;
                qrcb->inact_timer = rad_inact_timeout;
                qrcb->long_timer = 60;
                qrcb->last_base = -1;
                qrcb->last_speed = -1;
        return(&rcbs[address]);
    }
    return(NULL);
}

/*
    procdure: rtc_find

    author: Marvin Sojka

    created: 2/29/87

    called by:
        rtc_main
```

```
        rtc_in

    calls:
        none

    history:

    purpose: This is routine will locate a rcb with a given address
if the
             rcb exists.  If the rcb exists, the address of the rcb
is
             returned. Otherwise NULL is returned.

*/
rcb_ptr rcb_find(address)
WORD address;
{
        if (address >= MAX_TERMS)
                return(NULL);
    else if (rcbs[address].free == 1)
       return(&rcbs[address]);
    else
                return(NULL);
}

/*
    procdure: rtc_free

    author: Marvin Sojka

    created: 2/29/87

    called by:
        rtc_main
        rtc_in
        rtc_reset

    calls:
        none

    history:

    purpose: This is frees a rcb for other uses.

*/
void rcb_free(xrcb)
rcb_ptr xrcb;
{
    buff_free(xrcb->txq);
    buff_free(xrcb->rxq);
    xrcb->free = 0;
 }


/*
    procdure: rtc_init
```

```
    author: Marvin Sojka

    created: 2/29/87

    called by:
        main

    calls:
        none

    history:

    purpose: This is routine initialize the whole of the rcb's.avail
able.

*/
void rcb_init()
{
    WORD i;

    for (i=0;i<MAX_TERMS;i++)
        {
        rcbs[i].free = 0;
        rcbs[i].txq = NULL;
        rcbs[i].rxq = NULL;
        }
}
```

```
/*************************************************************************
****

Copyright (C)1991, Norand Corporation.  All Rights Reserved.


Revision History:

** $Log:   N:/3250/rtc/src/rad_stat.c_v  $
 *
 *    Rev 1.6   27 Feb 1992 16:06:20   SOJKA
 * Dual Host Support Release and RS-422 Bug
 *
 *    Rev 1.5   18 Feb 1992 10:48:00   SOJKA
 * UIF cosmetic fixes V0.23
 *
 *    Rev 1.4   31 Dec 1991 11:23:12   sojka
 * Complied with Library V0.21.
 *
 *    Rev 1.3   31 Dec 1991 10:41:00   sojka
 * Complied with Library V0.21.
 *
 *    Rev 1.2   20 Nov 1991 18:37:56   sojka
 * Added Copyright Information and PVCS Log keyword.

*************************************************************************
****/



#include "main.h"


void stats_init()
{

        memset(term_out_blocks,0,MAX_TERMS*sizeof(LONG));
        memset(term_in_blocks,0,MAX_TERMS*sizeof(LONG));
        memset(term_out_timeouts,0,MAX_TERMS*sizeof(LONG));
        memset(term_in_timeouts,0,MAX_TERMS*sizeof(LONG));

        memset(channel_out_blocks,0,4*sizeof(LONG));
        memset(channel_in_blocks,0,4*sizeof(LONG));
        memset(channel_out_timeouts,0,4*sizeof(LONG));
        memset(channel_in_timeouts,0,4*sizeof(LONG));
        memset(channel_interference,0,4*sizeof(LONG));
        memset(channel_collisions,0,4*sizeof(LONG));

        glob_out_blocks=0;
        glob_in_blocks=0;
        glob_out_timeouts=0;
        glob_in_timeouts=0;
        glob_interference=0;
        glob_collisions=0;
}
```

```
        rad_callsign_timer--;
    for (i=0;i<MAX_TERMS;i++)
    {
        qrcb = &rcbs[i];
        if (qrcb->free == RCB_INUSE)
            {
            if (qrcb->inact_timer)
                    {
                        qrcb->inact_timer--;
                        if ((qrcb->inact_timer == 0) &&
                                (qrcb->txq != NULL) &&
                                (qrcb->txq->type == RF_ECHO))
                        {
                                rtc_reject_echo(qrcb);
                        }
                    }

            if (qrcb->long_timer)
                        qrcb->long_timer--;
        }
    }
}

/*
    procdure: rad_timer_init

    author: Marvin Sojka

    created: 2/29/87

    history:
                7/11/90 mls modified for 68302

    called by:
        main

    calls:
        none

    purpose: This routine sets up the timer interrupt and the 80186
timer.

*/
BYTE rad_timer_tag[] = "RITX";

void rad_timer_init()
{
        LONG status;

        sc_gtime = (LONG *)0x440;

        /* initialize the soft timers */
    rad_timer = 0;
        rad_inact_timeout = 3;

        /* initialize the timers */
```

```
        set_interrupt_amx(INT_TIMER2,(LONG)timer_interrupt,&timer_in
t);
        ajdi();
        TIMER2_PTR->trr = (WORD) (CLOCK_SPEED/1000);
        TIMER2_PTR->tmr = TIMER_RST +
                                        TIMER_CLOCK_MASTER +
                                        TIMER_FRR +
                                        TIMER_ORI;
        TIMER2_PTR->ter = TIMER_EVENT_REF+TIMER_EVENT_CAP;
        ISR_PTR = INT_TIMER2_BIT;
        IPR_PTR = INT_TIMER2_BIT;
        IMR_PTR |= INT_TIMER2_BIT;
        ajei();

        /* create interval timer */
        rad_timer_tag[3] = '1' + rad_cfgx->port;
        status = ajtmcre(&rad_inactid,TEN_SECONDS,rad_inact,
                                        (struct tuser *)get_a5(),(ch
ar *)rad_timer_tag);
        if (status != 0)
                merr_write("RTC %d RAD_TIMER_INIT FAILED %ld\r",
                                rad_cfgx->port,status);

        ajtmwr(rad_inactid,TEN_SECONDS);

}
```

```
/*****************************************************************************
****

Copyright (C)1991, Norand Corporation.  All Rights Reserved.


Revision History:

** $Log:   N:/3250/rtc/src/rtc_acti.c_v  $
 *
 *    Rev 1.6   27 Feb 1992 16:06:16   SOJKA
 * Dual Host Support Release and RS-422 Bug
 *
 *    Rev 1.5   18 Feb 1992 10:47:56   SOJKA
 * UIF cosmetic fixes V0.23
 *
 *    Rev 1.4   31 Dec 1991 11:23:08   sojka
 * Complied with Library V0.21.
 *
 *    Rev 1.3   31 Dec 1991 10:40:56   sojka
 * Complied with Library V0.21.
 *
 *    Rev 1.2   20 Nov 1991 18:37:52   sojka
 * Added Copyright Information and PVCS Log keyword.

*****************************************************************************
****/



#include "main.h"

/*
    procdure: rtc_rcv_data

    author: Marvin Sojka

    created: 2/29/87

    called by:
        rtc_rx

    calls:
        none

    history:

    purpose: This is routine handles the placement of data from
             the rx_buffer into a temporary buffer for used by
             the system.

    returns: TERROR_EVENT if a chaining error is detected.
             D0_EVENT if no buffers are available.
             D1_EVENT if no length error has occurred.
             D2_EVENT if an length error has occurred.
```

```
*/
WORD rtc_rcv_data(command,length)
BYTE command;
WORD length;
{
    WORD event;

    /* check for a chain error */
    if ((crcb->rxq == NULL) &&
        (!(command & FIC)))
        event = TERROR_EVENT;
    else if ((crcb->rxq != NULL) &&
            (command & FIC))
        event = TERROR_EVENT;

    /* check for no buffers */
    else if (next_rx_buffer() == 0)
        event = D0_EVENT;

    else
    {
        /* setup the parameters */
        cbuff->chain_status = command & OIC;
                cbuff->type = RF_DATA;
        cbuff->address = crcb->address;
        event = D1_EVENT;
        cbuff->length = length;
    }
    return(event);
}

/*
    procdure: rtc_accept_data

    author: Marvin Sojka

    created: 2/29/87

    called by:
        rtc_norm_state

    calls:
        none

    history:

    purpose: This is handles queuing data that was created by rtc_rc
v_data to
             the rcb.  If buffer is LIC then the data is sent on.

*/
WORD rtc_accept_data()
{
    buff_ptr temp1;
```

```
        crcb->last_base = base;
        crcb->last_speed = scc_baud[base];

        term_in_blocks[crcb->address]++;
        channel_in_blocks[base]++;
        glob_in_blocks++;

    crcb->rx_length += cbuff->length;
    if (crcb->rxq == NULL)
        crcb->rxq=cbuff;
    else
    {
        temp1 = crcb->rxq;
        while (temp1->next != NULL)
            temp1 = temp1->next;
        temp1->next = cbuff;
    }
    cbuff->next = NULL;

    if (cbuff->chain_status & LIC)
    {
                crcb->rxq->from = (queue_ptr)crcb->time;
        rtc_data_indication(crcb->rxq);
        crcb->rx_length = 0;
        crcb->rxq = NULL;
    }
}
void rtc_accept_special()
{
        crcb->last_base = base;
        crcb->last_speed = scc_baud[base];

        term_in_blocks[crcb->address]++;
        channel_in_blocks[base]++;
        glob_in_blocks++;

        if (crcb->txq->type == RF_ECHO)
        {
                cbuff->type = RF_ECHO;
                post_main(cbuff,crcb->txq->from);
        }
        else
                rtc_data_indication(cbuff);
}
/*
        rtc_reject_echo
        sent a negative response to a echo command
*/
void rtc_reject_echo(qrcb)
rcb_ptr qrcb;
{
        buff_ptr temp;

        temp = qrcb->txq;
        qrcb->txq = temp->next;
        temp->next = NULL;
```

```
        temp->length = 0;
        post_main(temp,temp->from);
}

/*
    procdure: rtc_rcv_test

    author: Marvin Sojka

    created: 2/29/87

    called by:
        rtc_rx

    calls:
        none

    history:

    purpose: This is routine handles the input of a test frame from
the
             terminal.

    returns: D0_EVENT if no buffers are available.
             TEST_EVENT if no length error has occurred.
             D2_EVENT if an length error has occurred.

*/
rtc_rcv_test(length)
WORD length;
{
    WORD event,i;
    BYTE *data;
    BYTE *xdata;

        /* check for buffer */
    if (next_rx_buffer() == 0)
        event = D0_EVENT;
    else
    {
                /* get buffer to stuff data into */
        cbuff->chain_status = OIC;
        cbuff->type = RF_TEST;
        cbuff->address = crcb->address;
        event = TEST_EVENT;
        cbuff->length = length;
    }
    return(event);
}
/*
    procdure: rtc_rcv_version

    author: Marvin Sojka

    created: 2/29/87
```

```
    called by:
        rtc_rx

    calls:
        none

    history:

    purpose: This is routine handles the input of a version frame fr
om the
             terminal.

    returns: D0_EVENT if no buffers are available.
             VERSION_EVENT if no length error has occurred.
             D2_EVENT if an length error has occurred.

*/
rtc_rcv_version(length)
WORD length;
{
    WORD event;

        /* check for buffers available */
    if (next_rx_buffer() == 0)
        event = D0_EVENT;
    else
    {
                /* get buffer to stuff data into */
        cbuff->chain_status = OIC;
        cbuff->type = RF_VERSION;
        cbuff->address = crcb->address;
        event = VERSION_EVENT;
        cbuff->length = length;
    }
    return(event);
}


/*
    procdure: rtc_reset

    author: Marvin Sojka

    created: 2/29/87

    called by:
        rtc_main

    calls:
        rcb_free
        rcb_alloc

    history:

    purpose: This is routine handles the reset of a rcb into a initi
al state.
```

```
*/
void rtc_reset(qrcb)
rcb_ptr qrcb;
{
        buff_ptr temp,temp1;

        temp = qrcb->txq;
        temp1 = NULL;
        while (temp)
        {
                if (temp->type == RF_ECHO)
                {
                        temp1->next = temp;
                        temp1 = temp->next;
                        temp->next = NULL;
                        temp->length = 0;
                        post_main(temp,temp->from);
                        temp = temp1;
                }
                else
                {
                        temp1 = temp;
                        temp = temp->next;
                }
        }

    buff_free(qrcb->txq);
    buff_free(qrcb->rxq);
    qrcb->state = RESET_STATE;
    qrcb->vr = 0;
    qrcb->vs = 0;
    qrcb->txq = NULL;
    qrcb->rxq = NULL;
    qrcb->rx_length = 0;
        qrcb->last_base = base;
        qrcb->last_speed = scc_baud[base];
}
/*
    procdure: rtc_done

    author: Marvin Sojka

    created: 2/29/87

    called by:
        rtc_main

    calls:
        rtc_rsp_state
        rtc_version_state
        rtc_test_state

    history:

    purpose: This is routine handles the completion of a request fro
```

```
m the
                . host.  If type == 1 then a RF_DONE message is sent to t
he host.

*/
void rtc_done(type)
{
   buff_ptr temp,temp1;

   temp = crcb->txq;
   crcb->txq = crcb->txq->next;
   temp->next = NULL;

   term_out_blocks[crcb->address]++;
   channel_out_blocks[base]++;
   glob_out_blocks++;

   if (type)
   {
       temp->address = crcb->address;
       temp->chain_status = OIC;
       temp->type = RF_DONE;
       send_main(temp);
   }
   else
       buff_free(temp);
}
```

```
/*****************************************************************************
****

Copyright (C)1991, Norand Corporation.  All Rights Reserved.

Revision History:

** $Log:   N:/3250/rtc/src/rtc_idle.c_v  $
 *
 *    Rev 1.6   27 Feb 1992 16:05:54   SOJKA
 * Dual Host Support Release and RS-422 Bug
 *
 *    Rev 1.5   18 Feb 1992 10:47:32   SOJKA
 * UIF cosmetic fixes V0.23
 *
 *    Rev 1.4   31 Dec 1991 11:22:46   sojka
 * Complied with Library V0.21.
 *
 *    Rev 1.3   31 Dec 1991 10:40:36   sojka
 * Complied with Library V0.21.
 *
 *    Rev 1.2   20 Nov 1991 18:37:28   sojka
 * Added Copyright Information and PVCS Log keyword.

*****************************************************************************
****/



#include "main.h"


/*
    module:   rtc_idle.c

    procedure: rtc_idle

    author: Marvin Sojka

    created: 2/29/87

    called by:
        rtc_main.

    calls:
        none.

    history:

    purpose: this routine is called the first thing after a up comma
nd
             for each device in the up_q.  This routine will handlin
g
             setting the event that is ocurring from the idle state.
```

```
    returns:
            IDLE0_EVENT -- nothing on crcb->txq queue.
            IDLE1_EVENT -- data on crcb->txq and LIC.
            IDLE2_EVENT -- data on crcb->txq and ~LIC.
            IDLE3_EVENT -- test frame on crcb->txq
            IDLE4_EVENT -- version frame on crcb->txq.
            TIMEOUT_EVENT -- invalid data on crcb->txq.
*/

rtc_idle()
{
        buff_ptr temp;
    WORD event;


        /* if no buffer then IDLE0_EVENT */
    if (crcb->txq == NULL)
        event = IDLE0_EVENT;
        else if (crcb->state == ECHO_STATE)
                event = TIMEOUT_EVENT;
    else
    {
                /* check Type for IDLEX_EVENT Type */
        switch (crcb->txq->type)
        {
            case RF_DATA:
                                /* Check to see if Last in Chain */
                if (crcb->txq->chain_status & LIC)
                    event = IDLE1_EVENT;
                else
                    event = IDLE2_EVENT;
                break;

                        case RF_ECHO:
            case RF_TEST:
                event = IDLE3_EVENT;
                break;

            case RF_VERSION:
                event = IDLE4_EVENT;
                break;

            default:
                event = TIMEOUT_EVENT;
                break;
        }
    }
    return(event);
}
```

```
/*****************************************************************
****

Copyright (C)1991, Norand Corporation.  All Rights Reserved.


Revision History:

** $Log:   N:/3250/rtc/src/rtc_ind.c_v  $
 *
 *    Rev 1.6   27 Feb 1992 16:05:38   SOJKA
 * Dual Host Support Release and RS-422 Bug
 *
 *    Rev 1.5   18 Feb 1992 10:47:16   SOJKA
 * UIF cosmetic fixes V0.23
 *
 *    Rev 1.4   31 Dec 1991 11:22:28   sojka
 * Complied with Library V0.21.
 *
 *    Rev 1.3   31 Dec 1991 10:40:20   sojka
 * Complied with Library V0.21.
 *
 *    Rev 1.2   20 Nov 1991 18:37:14   sojka
 * Added Copyright Information and PVCS Log keyword.

*****************************************************************
****/



#include "main.h"


/*
    procdure: rtc_reset_indication

    author: Marvin Sojka

    created: 2/29/87

    called by:
        rtc_main

    calls:
        none

    history:

    purpose: This is creates a RF_REBOOT buffer for use by the host.

*/
rtc_reset_indication()
{
    buff_ptr temp,i;
        /* if sna is not active and there is nothing on the txq then
```

```
                send a reset to the upper layers.
        */
        temp = buff_get();
        temp->address = crcb->address;
        temp->chain_status = OIC;
        temp->type = RF_REBOOT;
        temp->length = 1;
        temp->data[0] = crcb->type;

        send_main(temp);

}

/*
    procdure: rtc_data_indication

    author: Marvin Sojka

    created: 2/29/87

    called by:
        rtc_accept_data
        rtc_version_state
        rtc_test_state

    calls:
        none

    history:

    purpose: This is creates a passes data buffers to the HOST.

*/
rtc_data_indication(nbuff)
buff_ptr nbuff;
{
    send_main(nbuff);
}
```

```
/*************************************************************************
****

Copyright (C)1991, Norand Corporation.  All Rights Reserved.


Revision History:

** $Log:   N:/3250/rtc/src/rtc_init.c_v  $
 *
 *    Rev 1.6   27 Feb 1992 16:05:40   SOJKA
 * Dual Host Support Release and RS-422 Bug
 *
 *    Rev 1.5   18 Feb 1992 10:47:18   SOJKA
 * UIF cosmetic fixes V0.23
 *
 *    Rev 1.4   31 Dec 1991 11:22:32   sojka
 * Complied with Library V0.21.
 *
 *    Rev 1.3   31 Dec 1991 10:40:22   sojka
 * Complied with Library V0.21.
 *
 *    Rev 1.2   20 Nov 1991 18:37:16   sojka
 * Added Copyright Information and PVCS Log keyword.

*************************************************************************
****/



#include "main.h"


WORD scc_baud[] = {BAUD4800,BAUD4800,BAUD4800,BAUD4800}; /* index by
 base */
WORD up_inactive_count[] = {0,0,0,0};
  /* index by base */
WORD fup_inactive_count[] = {0,0,0,0};
  /* index by base */
WORD slot[] = {1,1,1,1};
                  /* index by base */
WORD fslot[] = {1,1,1,1};
                  /* index by base */


void rtc_init()
{

        rtc_rx_buff[0] = buff_get();
        rtc_rx_buff[1] = buff_get();

        /* disable squelch checking in assembly routines */
        squelch_check[0] = 0;
        squelch_check[1] = 0;
```

```
        port                            = 0;
        base                            = 0;

        rtc_rxx[0]                      = 0;
        rtc_rxx[1]                      = 0;
        rtc_slots                       = RTC_MIN_SLOTS;

}
```

```
/*****************************************************************************
****

Copyright (C)1991, Norand Corporation.  All Rights Reserved.


Revision History:

** $Log:   N:/3250/rtc/src/rtc_main.c_v  $
 *
 *    Rev 1.6   27 Feb 1992 16:05:34   SOJKA
 * Dual Host Support Release and RS-422 Bug
 *
 *    Rev 1.5   18 Feb 1992 10:47:12   SOJKA
 * UIF cosmetic fixes V0.23
 *
 *    Rev 1.4   31 Dec 1991 11:22:22   sojka
 * Complied with Library V0.21.
 *
 *    Rev 1.3   31 Dec 1991 10:40:14   sojka
 * Complied with Library V0.21.
 *
 *    Rev 1.2   20 Nov 1991 18:37:08   sojka
 * Added Copyright Information and PVCS Log keyword.

*****************************************************************************
****/



#include "main.h"

LONG *sc_gtime;

#define NULL_DELAY                      5
#define WAIT_DELAY                      20
/*
    module:  rtc_main.c

    procdure: rtc_main

    author: Marvin Sojka

    created: 2/29/87

    history:

    purpose: main loop for rtc primary protocol.  The sequence is si
mple.
             rtc_main first calls rtc_switch to decide on speed and
port.
             rtc_main will then call rtc_up to send a unnumbered pol
l.
             The responding devices are in up_q and the count is in
             up_q_count.  Each entry in up_q is then polled until no
```

```
thing is
                left to communicate to it for.  It the terminal has jus
t been
                reseted or connected then a corresponding indication is
 sent to
                the higher layers.
*/

void rtc_main()
{
    WORD event,i,j;

        rtc_switch();

    /* send a unnumbered poll and see if any devices are out there *
/
    rtc_up();

    /* check each member of the up_q for activity */
    for (j=0;j<up_q_count;j++)
    {
                /* make sure the device is defined */
        if ((crcb = rcb_find(up_q[j] & 0x7F)) ||
            (crcb = rcb_alloc(up_q[j] & 0x7F)))
        {
                        lcd_addr(crcb->address);

                        if (crcb->rxq == NULL)
                                crcb->time = *sc_gtime;
                        crcb->retrys = 0;
                        if (up_q[j] & 0x80)
                                crcb->retry_max = 1;
                        else
                                crcb->retry_max = 3;
            /* get idle state event */
            event = rtc_idle();
            /* run event thru state machine */
            while ((i = rtc_prifsm(event))== NOT_DONE)
                event = rtc_receive();

                }
        }
}

void rtc_rx_start0()
{
    /* clear rts */
    ptt_off(port);

        /* set delay for rtc_rx_initialization routine */
        rad_timer = NULL_DELAY;
        rad_timer_routine = rtc_rx_start1;
}

void rtc_rx_start1()
```

```
{

        if (rad_cfgx->remote_base[base] == REMOTE_ON)
                rad_timer = WAIT_DELAY + rad_cfgx->remote_offset[bas
e];
        else
                rad_timer = WAIT_DELAY;
        rad_timer_routine = rad_timeout;
        if (rad_cfgx->base == MBA)
        {
                if ((rad_cfgx->mba[base] & PORTA) == PORTA)
                        rtc_rx_init(PORT0);
                if ((rad_cfgx->mba[base] & PORTB) == PORTB)
                        rtc_rx_init(PORT1);
        }
        else
                rtc_rx_init(port);
}

/*
        rtc_receive
        delays RX_DELAY time
        starts rtc_rx
        wait for block_delay_timeout
*/
WORD rtc_receive()
{

        WORD event;

        /* wait for receive value */
        event = pend_mailbox(0);

        /* if timeout leave as is else check receive buffer */
        if (event == TX_TIMEOUT_EVENT)
        {
                event = TIMEOUT_EVENT;
        }
        else if (event != TIMEOUT_EVENT)
        {
                if (rad_cfgx->base == MBA)
                {
                        if ((rad_cfgx->mba[base] & MBA_MASK) == PORT
AB)
                        {
                                if ((event = rtc_rx(0)) == (WORD)-1)
                                        event = rtc_rx(1);
                        }
                        else if ((rad_cfgx->mba[base] & MBA_MASK) ==
 PORTA)
                                        event = rtc_rx(0);
                        else if ((rad_cfgx->mba[base] & MBA_MASK) ==
 PORTB)
                                        event = rtc_rx(1);
                }
                else
```

```
                        event = rtc_rx(port);

                /* receive error is a timeout */
                if (event != TIMEOUT_EVENT)
                {
                        lcd_rx_status(LCD_RX_GOOD);
                        crcb->inact_timer = rad_inact_timeout;
                        crcb->long_timer = 60;
                }
                else
                {
                        lcd_rx_status(LCD_RX_CRC_ERROR);
                        if (crcb->txq != NULL)
                        {
                                term_out_timeouts[crcb->address]++;
                                channel_out_timeouts[base]++;
                                glob_out_timeouts++;
                        }
                        else
                        {
                                term_in_timeouts[crcb->address]++;
                                channel_in_timeouts[base]++;
                                glob_in_timeouts++;
                        }
                }
        }
        else
        {
                lcd_rx_status(LCD_RX_TIMEOUT);

                crcb->retrys++;
                if (crcb->retrys < crcb->retry_max)
                        event = rtc_idle();
                else
                {
                        crcb->last_base = -1;
                        event = TIMEOUT_EVENT;
                }
                if (crcb->txq != NULL)
                {
                        term_out_timeouts[crcb->address]++;
                        channel_out_timeouts[base]++;
                        glob_out_timeouts++;
                }
                else
                {
                        term_in_timeouts[crcb->address]++;
                        channel_in_timeouts[base]++;
                        glob_in_timeouts++;
                }
        }
        return(event);
}


/*
```

```
    procdure: rad_timeout

    author: Marvin Sojka

    created: 2/29/87

    history:

    called by:
        rtc_event_timer

    calls:
        none.

    purpose: Sends a TIMEOUT_EVENT message to mailbox when the
             rf_timer expires and rf_timer_routine is set to rtc_tim
eout.

*/
void rad_timeout()
{
        if (rad_cfgx->base == MBA)
        {
                if ((rad_cfgx->mba[base] & PORTA) == PORTA)
                        rad_stop_scc(PORT0);
                if ((rad_cfgx->mba[base] & PORTB) == PORTB)
                        rad_stop_scc(PORT1);
        }
        else
                rad_stop_scc(port);
        post_mailbox(TIMEOUT_EVENT);
}
```

```
/*************************************************************************
****

Copyright (C)1991, Norand Corporation.  All Rights Reserved.


Revision History:

** $Log:   N:/3250/rtc/src/rtc_pri.c_v  $
 *
 *    Rev 1.6   27 Feb 1992 16:05:12   SOJKA
 * Dual Host Support Release and RS-422 Bug
 *
 *    Rev 1.5   18 Feb 1992 10:46:52   SOJKA
 * UIF cosmetic fixes V0.23
 *
 *    Rev 1.4   31 Dec 1991 11:22:00   sojka
 * Complied with Library V0.21.
 *
 *    Rev 1.3   31 Dec 1991 10:39:52   sojka
 * Complied with Library V0.21.
 *
 *    Rev 1.2   20 Nov 1991 18:36:44   sojka
 * Added Copyright Information and PVCS Log keyword.

*************************************************************************
****/



#include "main.h"

/*
    module:  rtc_pri.c

    procdure: rtc_prifsm

    author: Marvin Sojka

    created: 2/29/87

    history:

    purpose: handle main finite state machin for rtc protocol.
                        see RTC overveiw documentation for the prim
ary
                        Finite state machine.

*/

WORD rtc_prifsm(event)
{
    WORD retcode;

    switch (crcb->state)
```

```
    {
        case RESET_STATE:
            retcode = rtc_reset_state(event);
            break;

        case NORM_STATE:
            retcode = rtc_norm_state(event);
            break;

        case RSP_STATE:
            retcode = rtc_rsp_state(event);
            break;

        case TEST_STATE:
            retcode = rtc_test_state(event);
            break;

        case VERSION_STATE:
            retcode = rtc_version_state(event);
            break;

        }
        return(retcode);
}

WORD rtc_reset_state(event)
WORD event;
{

    switch (event)
    {
          case IDLE0_EVENT:
          case IDLE1_EVENT:
          case IDLE2_EVENT:
          case IDLE3_EVENT:
          case IDLE4_EVENT:
                  case IDLE5_EVENT:
            rtc_send_reset();
            return(NOT_DONE);
            break;

         case TIMEOUT_EVENT:
             return(DONE);
             break;

         case RESET_EVENT:
             rtc_reset_indication();
             crcb->state = NORM_STATE;
             crcb->vr = crcb->vs = 0;
             return(CONNECT_CODE);

         case TEST_EVENT:
         case VERSION_EVENT:
         case D1_EVENT:
             buff_free(cbuff);
```

```
        case ACK0_EVENT:
        case ACK1_EVENT:
        case ACK2_EVENT:
        case ACKN0_EVENT:
        case ACKN1_EVENT:
        case ACKN2_EVENT:
        case D0_EVENT:
        case D2_EVENT:
        case TERROR_EVENT:
                        rtc_reset(crcb);
           return(DONE);
    }
}

WORD rtc_norm_state(event)
WORD event;
{

    switch (event)
    {

          case IDLE0_EVENT:
              rtc_send_poll();
              return(NOT_DONE);

          case IDLE1_EVENT:
          case IDLE2_EVENT:
              crcb->vs = (crcb->vs + 1) & 0x01;
              rtc_send_data();
              crcb->state = RSP_STATE;
              return(NOT_DONE);

          case IDLE3_EVENT:
              rtc_send_test();
              crcb->state = TEST_STATE;
              return(NOT_DONE);

          case IDLE4_EVENT:
              rtc_send_version();
              crcb->state = VERSION_STATE;
              return(NOT_DONE);

                  case IDLE5_EVENT:
              rtc_send_test();
              crcb->state = ECHO_STATE;
              return(NOT_DONE);

          case TIMEOUT_EVENT:
              return(DONE);

          case RESET_EVENT:
                        rtc_reset(crcb);
              return(DONE);

          case TEST_EVENT:
          case VERSION_EVENT:
```

```
            buff_free(cbuff);
                        rtc_reset(crcb);
            return(DONE);

        case ACK0_EVENT:
        case ACK1_EVENT:
            return(DONE);

        case ACK2_EVENT:
                        rtc_reset(crcb);
            return(DONE);

        case ACKN0_EVENT:
        case ACKN1_EVENT:
            return(DONE);

        case ACKN2_EVENT:
                        rtc_reset(crcb);
            return(DONE);

        case D0_EVENT:
            return(DONE);

        case D1_EVENT:
            rtc_accept_data();
            crcb->vr = (crcb->vr+1) & 0x01;
            rtc_send_poll();
            return(NOT_DONE);

        case D2_EVENT:
        case TERROR_EVENT:
                        rtc_reset(crcb);
            return(DONE);
    }
}

WORD rtc_rsp_state(event)
WORD event;
{
    switch (event)
    {
        case IDLE0_EVENT:
        case IDLE1_EVENT:
        case IDLE2_EVENT:
            rtc_send_npoll();
            return(NOT_DONE);

        case IDLE3_EVENT:
        case IDLE4_EVENT:
                case IDLE5_EVENT:
                        rtc_reset(crcb);
            return(DONE);

        case TIMEOUT_EVENT:
            return(DONE);
```

```
            case RESET_EVENT:
                            rtc_reset(crcb);
                return(DONE);

            case ACK0_EVENT:
                rtc_done(1);
                crcb->state = NORM_STATE;
                return(DONE);

            case ACK1_EVENT:
                crcb->vs = (crcb->vs + 1) & 0x01;
                rtc_done(0);
                rtc_send_data();
                return(NOT_DONE);

            case ACK2_EVENT:
                rtc_send_data();
                return(NOT_DONE);

            case ACKN0_EVENT:
                rtc_done(1);
                crcb->state = NORM_STATE;
                return(DONE);

            case ACKN1_EVENT:
                rtc_done(0);
                return(DONE);

            case ACKN2_EVENT:
                crcb->vs = (crcb->vs + 1) & 0x01;
                return(DONE);

            case TEST_EVENT:
            case VERSION_EVENT:
            case D1_EVENT:
               buff_free(cbuff);
            case D0_EVENT:
            case D2_EVENT:
            case TERROR_EVENT:
                            rtc_reset(crcb);
                return(DONE);

        }
}

WORD rtc_test_state(event)
WORD event;
{

    switch (event)
    {
          case IDLE0_EVENT:
          case IDLE1_EVENT:
          case IDLE2_EVENT:
          case IDLE4_EVENT:
             .     case IDLE5_EVENT:
```

```
                        rtc_reset(crcb);
            return(DONE);

        case IDLE3_EVENT:
            rtc_send_test();
            crcb->state = TEST_STATE;
            return(NOT_DONE);

        case TIMEOUT_EVENT:
            crcb->state = NORM_STATE;
            return(DONE);

        case RESET_EVENT:
                        rtc_reset(crcb);
            return(DONE);

        case TEST_EVENT:
                        rtc_accept_special();
            rtc_done(0);
            crcb->state = NORM_STATE;
            return(DONE);

        case D1_EVENT:
        case VERSION_EVENT:
            buff_free(cbuff);

        case ACK0_EVENT:
        case ACK1_EVENT:
        case ACK2_EVENT:
        case ACKN0_EVENT:
        case ACKN1_EVENT:
        case ACKN2_EVENT:
        case D0_EVENT:
        case D2_EVENT:
        case TERROR_EVENT:
                        rtc_reset(crcb);
            return(DONE);
    }
}


WORD rtc_version_state(event)
WORD event;
{

    switch (event)
    {
        case IDLE0_EVENT:
        case IDLE1_EVENT:
        case IDLE2_EVENT:
        case IDLE3_EVENT:
                case IDLE5_EVENT:
                        rtc_reset(crcb);
            return(DONE);

        case IDLE4_EVENT:
```

```
                rtc_send_version();
                crcb->state = VERSION_STATE;
                return(NOT_DONE);

            case TIMEOUT_EVENT:
                crcb->state = NORM_STATE;
                return(DONE);

            case RESET_EVENT:
                            rtc_reset(crcb);
                return(DONE);

            case VERSION_EVENT:
                            rtc_accept_special();
                rtc_done(0);
                crcb->state = NORM_STATE;
                return(DONE);

            case D1_EVENT:
            case TEST_EVENT:
                buff_free(cbuff);

            case ACK0_EVENT:
            case ACK1_EVENT:
            case ACK2_EVENT:
            case ACKN0_EVENT:
            case ACKN1_EVENT:
            case ACKN2_EVENT:
            case D0_EVENT:
            case D2_EVENT:
            case TERROR_EVENT:
                            rtc_reset(crcb);
                return(DONE);
        }
    }
```

```
/*****************************************************************
****

Copyright (C)1991, Norand Corporation.  All Rights Reserved.


Revision History:

** $Log:   N:/3250/rtc/src/rtc_rx.c_v  $
 *
 *    Rev 1.6   27 Feb 1992 16:05:52   SOJKA
 * Dual Host Support Release and RS-422 Bug
 *
 *    Rev 1.5   18 Feb 1992 10:47:30   SOJKA
 * UIF cosmetic fixes V0.23
 *
 *    Rev 1.4   31 Dec 1991 11:22:42   sojka
 * Complied with Library V0.21.
 *
 *    Rev 1.3   31 Dec 1991 10:40:32   sojka
 * Complied with Library V0.21.
 *
 *    Rev 1.2   20 Nov 1991 18:37:26   sojka
 * Added Copyright Information and PVCS Log keyword.

*****************************************************************
****/



#include "main.h"


/*
    module:  rtc_rx.c

    procdure: rtc_rx

    author: Marvin Sojka

    created: 2/29/87

    history:

    purpose: this routine is used to decode the received data to
             determine which event occurred for the primary state ma
chine

    DATA LINK COMMAND          OTHER CONSIDERATIONS          EVENT
                                BAD CRC                      TIMEOUT_
EVENT
    RESET                                                    RESET_EV
ENT
    POLL0 or POLL1             GOOD ACK and NO DATA          ACK0_EVE
```

```
NT
    POLL0 or POLL1              GOOD ACK and DATA           ACK1_EVE
NT
    POLL0 or POLL1              BAD ACK                     ACK2_EVE
NT
    NPOLL0 or NPOLL1            GOOD ACK and NO DATA        ACKN0_EV
ENT
    NPOLL0 or NPOLL1            GOOD ACK and DATA           ACKN1_EV
ENT
    NPOLL0 or NPOLL1            BAD ACK                     ACKN2_EV
ENT
    DATA (see rtc_rcv_data)     Chain Error                 TERROR_E
VENT
    DATA                        NO BUFFER                   D0_EVENT
    DATA                        Buffer_OverFlow             D2_EVENT
    DATA                        OK                          D1_EVENT
    TEST (see rtc_rcv_test)     NO BUFFER                   D0_EVENT
    TEST                        OK                          TEST_EVE
NT
    VERSION(see rtc_rcv_version)NO BUFFER                   D0_EVENT
    VERSION                     OK                          VERSION_
EVENT
*/
WORD current_who;
WORD rtc_rx(who)
WORD who;
{

    WORD event;
        WORD i,j;

        cbuff = rtc_rx_buff[who];

        if (rtc_rxx[who]==0)
                return -1;

        current_who = who;

    switch (cbuff->command & 0x0F)
    {

                case RESET:
                        event = RESET_EVENT;
                        crcb->type = cbuff->data[0] & 0xFF;
                        break;

        case POLL0:
        case POLL1:
                        /* if NPOLL is set then the events are ACKNX
_EVENTS otherwise
                                        they are ACKX_EVENT */
            if (cbuff->command & NPOLL)
                event = ACKN0_EVENT;
            else
                event = ACK0_EVENT;
```

```
                        /*
                                        if vs value is good then check if la
st in chain or not
                        */
            if ((cbuff->command & 0x01) == crcb->vs)
            {
                                        if ((crcb->txq != NULL) &&
                                                (!(crcb->txq->chain_status &
 LIC)))
                                                event++;
            }
            else
                                        event += 2;
            break;

                case CDATA:
        case DATA:
            event = rtc_rcv_data(cbuff->command,
                                                                        rtc
_rx_length[who]-RTC_OVERHEAD);
            break;

        case TEST:
            event = rtc_rcv_test(rtc_rx_length[who]-RTC_OVERHEAD);
            break;

        case VERSION:
            event = rtc_rcv_version(rtc_rx_length[who]-RTC_OVERHEAD)
;
            break;

        default:
           event = TIMEOUT_EVENT;
           break;


    }

    return(event);

}

WORD next_rx_buffer()
{

        buff_ptr temp;

        if ((temp = buff_get()) == NULL)
                return 0;
        else
        {
                rtc_rx_buff[current_who] = temp;
                return 1;
        }
}
```

```
/*****************************************************************************
****

Copyright (C)1991, Norand Corporation.  All Rights Reserved.


Revision History:

** $Log:   N:/3250/rtc/src/rtc_send.c_v  $
 *
 *    Rev 1.6   27 Feb 1992 16:06:12   SOJKA
 * Dual Host Support Release and RS-422 Bug
 *
 *    Rev 1.5   18 Feb 1992 10:47:52   SOJKA
 * UIF cosmetic fixes V0.23
 *
 *    Rev 1.4   31 Dec 1991 11:23:04   sojka
 * Complied with Library V0.21.
 *
 *    Rev 1.3   31 Dec 1991 10:40:54   sojka
 * Complied with Library V0.21.
 *
 *    Rev 1.2   20 Nov 1991 18:37:48   sojka
 * Added Copyright Information and PVCS Log keyword.

*****************************************************************************
****/



#include "main.h"

void rtc_tx_start0();
void rtc_tx_start1();
void rtc_tx_squelch();
void rtc_rx_start0();

rtc_send_poll()
{
        lcd_tx_status(LCD_TX_POLL);

        if (crcb->vr)
                rtc_send(crcb->address,POLL1 | 0xC0,NULL,0);
        else
                rtc_send(crcb->address,POLL0 | 0xC0,NULL,0);
}

rtc_send_npoll()
{
        lcd_tx_status(LCD_TX_POLL);

        if (crcb->vr)
                rtc_send(crcb->address,POLL1 | 0xC0 | NPOLL,NULL,0);
        else
                rtc_send(crcb->address,POLL0 | 0xC0 | NPOLL,NULL,0);
```

```
}

rtc_send_reset()
{
        lcd_tx_status(LCD_TX_RESET);
        rtc_send(crcb->address,RESET | 0xC0 ,NULL,0);
}

rtc_send_data()
{
        buff_ptr temp;
        lcd_tx_status(LCD_TX_DATA);
        temp = crcb->txq;
        if (temp->type == RF_DATA)
                rtc_send(crcb->address,DATA | 0xC0 | temp->chain_sta
tus,
                                        &temp->data[0],temp->length);
        else
                rtc_send(crcb->address,CDATA | 0xC0 | temp->chain_st
atus,
                                        &temp->data[0],temp->length);
}

rtc_send_version()
{
        lcd_tx_status(LCD_TX_VERSION);
        rtc_send(crcb->address,VERSION | 0xC0 ,NULL,0);
}


rtc_send_test()
{
        buff_ptr temp;
        lcd_tx_status(LCD_TX_TEST);
        temp = crcb->txq;
        rtc_send(crcb->address,TEST | 0xC0,&temp->data[0],temp->leng
th);
}

rtc_send( address, command, data, length )
BYTE address, command;
BYTE *data;
WORD length;
{

        WORD i,nlength;
        BYTE *x;

        if (rad_cfgx->base == MBA)
                ptt(PORT0);
        else
                ptt(port);

        rad_timer_routine = rtc_tx_start0;

/*      if (crcb->type == CRICKET_SHIT)
```

```
                rad_timer = PTT_DELAY/2 + 20;
        else
                rad_timer = PTT_DELAY/2;
*/
        rad_timer = PTT_DELAY/2;
        rad_tx_buffer[0] = STX;
        rad_tx_buffer[1] = address | 0x80;
        rad_tx_buffer[2] = command;
        x = &rad_tx_buffer[3];
        rad_tx_length = 3;
        for (i=0;i<length;i++)
        {
                if ((*data == STX) ||
                        (*data == ETX) ||
                        (*data == DLE))
                {
                        rad_tx_length+=2;
                        *x++ = DLE;
                        *x++ = *data++ | 0x80;
                }
                else
                {
                        rad_tx_length++;
                        *x++ = *data++;
                }
        }
        rad_tx_buffer[rad_tx_length] = ETX;
        rad_tx_length++;

        i = calc_crc(&rad_tx_buffer[1],rad_tx_length-1);
        rad_tx_buffer[rad_tx_length++] = (BYTE)i;
        rad_tx_buffer[rad_tx_length++] = (BYTE)(i>>8);

}
void rtc_tx_start0()
{
        scc_send_one(port);
        rad_timer_routine = rtc_tx_start1;
        rad_timer = PTT_DELAY/2+1;
}
void rtc_tx_start1()
{
        rad_tx_routine = rtc_rx_start0;
        rad_timer = 0;
        rtc_tx_init(port);
}
```

```
/*****************************************************************************
****

Copyright (C)1991, Norand Corporation.  All Rights Reserved.


Revision History:

** $Log:   N:/3250/rtc/src/rtc_up.c_v  $
 *
 *    Rev 1.6   27 Feb 1992 16:06:00   SOJKA
 * Dual Host Support Release and RS-422 Bug
 *
 *    Rev 1.5   18 Feb 1992 10:47:40   SOJKA
 * UIF cosmetic fixes V0.23
 *
 *    Rev 1.4   31 Dec 1991 11:22:52   sojka
 * Complied with Library V0.21.
 *
 *    Rev 1.3   31 Dec 1991 10:40:42   sojka
 * Complied with Library V0.21.
 *
 *    Rev 1.2   20 Nov 1991 18:37:34   sojka
 * Added Copyright Information and PVCS Log keyword.

*****************************************************************************
****/



#include "main.h"

#define SQUELCH_WAIT    10
#define SQUELCH_TIMEOUT 250

WORD squelch_count;
WORD squelch_timeout;
WORD squelch_first;
WORD squelch_bad;

WORD cslot;                                     /* global slot and c
ollisions indications */
WORD collisions;

/*
        rtc_up
        This routine
*/
void rtc_up()
{
        WORD done,i,j;
        rcb_ptr xrcb;

        done = 0;
        /*
```

```
                loop till exit required
        */

        if (scc_baud[base] == BAUD4800)
                rtc_slots = slot[base];
        else
                rtc_slots = fslot[base];

        up_flag = 1;

        /* reset counters */
        collisions = 0;
        up_q_count = 0;
        cslot = 0;

        /* send UP command on RF Link */
        rtc_send_up(rtc_slots);
        pend_mailbox(0);

        /* if count is greater than 0
           check if no collisions and rtc_slots is greater than
           RTC_SLOT_COUNT over RTC_MIN_SLOTS then decrement rtc_slot
s counter
        */
        if (i == TX_TIMEOUT_EVENT)
        {
                up_flag = 0;
                up_q_count = 0;
                return;
        }
        else if (up_q_count > 0)
        {
                if ( (rtc_slots > RTC_MIN_SLOTS+RTC_SLOT_COUNT) &&
                        (!(collisions))
                   )
                        rtc_slots --;
        }
        /*
                if collisions were detected then
                increment by rtc_slots by RTC_SLOT_COUNT to
                allow for more units to response
        */
        else if (collisions)
        {
                rtc_slots += RTC_SLOT_COUNT;
                /*
                        if rtc_slots reaches threshold then
                        decement rtc_slots and queue all
                        active terminals
                */
                if (rtc_slots > RTC_MAX_SLOTS)
                {
                        rtc_slots = RTC_MIN_SLOTS;
                        for (i=0;i<MAX_TERMS;i++)
                        {
                                xrcb = &rcbs[i];
```

```
                                        if (xrcb->free == RCB_INUSE)
                                        {
                                                up_q[up_q_count] = xrcb->add
ress | 0x80;
                                                up_q_count++;
                                        }
                                }
                        }
                }
                /*
                        otherwise decrement rtc_slots by one
                */
                else
                {
                        if (rtc_slots != RTC_MIN_SLOTS)
                                rtc_slots --;
                }
                /* save slots in apporiate manner */
                if (scc_baud[base] == BAUD4800)
                        slot[base] = rtc_slots;
                else
                        fslot[base] = rtc_slots;

                up_flag = 0;
}

/*
        rtc_send_up
        sends up with slots as passed.
        also includes all terminals that data for
        terminals that are inactive
*/

rtc_send_up(slots)
WORD slots;
{
        WORD i,j;
        rcb_ptr xrcb;

        lcd_tx_status(LCD_TX_UP);
        lcd_heartbeats(slots);

        /* turn on ptt line */
        squelch_timeout = SQUELCH_TIMEOUT;
        squelch_count = SQUELCH_WAIT;
        squelch_first = 1;
        rtc_up_squelch();

        /* setup stanard buffer */
        rad_tx_buffer[0] = STX;
        rad_tx_buffer[1] = 0xFF;
        rad_tx_buffer[2] = UP;
        rad_tx_buffer[3] = slots | 0x80;
        rad_tx_length = 4;

        /*
```

```
                queue all active terminals with
                data to send to up_q
        */
        for (i=0;i<MAX_TERMS;i++)
        {
                xrcb = &rcbs[i];
                if (xrcb->free == RCB_INUSE)
                {
                        if ((xrcb->state == RESET_STATE) ||
                                (xrcb->txq != NULL))
                        {
                            if (xrcb->inact_timer == 0)
                                {
                                        rad_tx_buffer[rad_tx_length]
 = xrcb->address | 0x80;
                                        rad_tx_length++;
                                }
                                else if (xrcb->txq->type == RF_ECHO)
                                {
                                        if ((xrcb->txq->chain_status
 == (BYTE)-1) ||
                                            (xrcb->txq->chain_status
 == base))
                                        {
                                                up_q[up_q_count] = x
rcb->address | 0x80;
                                                up_q_count++;
                                        }
                                }
                                else if ((xrcb->last_base == (WORD)-
1) ||
                                          ((xrcb->last_base =
= base) &&
                                           (xrcb->last_speed
== scc_baud[base]))
                                        )
                                {
                                        up_q[up_q_count] = xrcb->add
ress | 0x80;
                                        up_q_count++;
                                }
                        }
                }
        }
        rad_tx_buffer[rad_tx_length] = ETX;
        rad_tx_length++;

        i = calc_crc(&rad_tx_buffer[1],rad_tx_length-1);
        rad_tx_buffer[rad_tx_length++] = (BYTE)i;
        rad_tx_buffer[rad_tx_length++] = (BYTE)(i>>8);

}

/*
        rtc_up_squelch
*/
```

```
void rtc_up_squelch()
{
        WORD i;
        if (rad_cfgx->base == MBA)
        {
                i = 0;
                if ((rad_cfgx->mba[base] & PORTA) == PORTA)
                        i |= check_squelch(PORT0);
                if ((rad_cfgx->mba[base] & PORTB) == PORTB)
                        i |= check_squelch(PORT1);
        }
        else
                i = check_squelch(port);

        if (i)
        {
                squelch_count = 0;
                if (squelch_first)
                {
                        lcd_tx_status(LCD_TX_INTERFERENCE);
                        channel_interference[base]++;
                        glob_interference++;
                        squelch_first = 0;
                }
        }
        else
        {
                if (squelch_count >= SQUELCH_WAIT)
                {
                        /* turn on ptt line */
                        if (rad_cfgx->base == MBA)
                                ptt(PORT0);
                        else
                                ptt(port);
                        /* call first step of send */
                        rad_timer_routine = rtc_up_start0;
                        rad_timer = PTT_DELAY/2;
                        squelch_timeout=0;
                        squelch_bad = 0;
                        lcd_sys_status(RAD_GOOD);
                        return;
                }
                else
                        squelch_count++;
        }

        /* decement squelch timeout counter */
        squelch_timeout--;
        /* if squelch is nonzero then
           setup for 1 ms timer to check for squelch again
           else timeout.
        */
        if (squelch_timeout)
        {
                rad_timer_routine = rtc_up_squelch;
                rad_timer = 1;
```

```
        }
        else
        {
                if (rad_cfgx->base == MBA)
                {
                        if ((rad_cfgx->mba[base] & PORTA) == PORTA)
                                rad_stop_scc(PORT0);
                        if ((rad_cfgx->mba[base] & PORTB) == PORTB)
                                rad_stop_scc(PORT1);
                }
                else
                        rad_stop_scc(port);
                squelch_bad++;
                if (squelch_bad >= SQUELCH_WAIT)
                                lcd_sys_status(RAD_RADIO);
                post_mailbox(TX_TIMEOUT_EVENT);
        }
}
/*
        rtc_up_start0
        This routine outputs a 0x0f to initialize
        receive detect mechanism in radio receivers.
        Sets up rad_timer to call routine to start txing.
*/
void rtc_up_start0()
{
        scc_send_one(port);
        rad_timer_routine = rtc_up_start1;
        rad_timer = PTT_DELAY/2+1;
}

/*
        rtc_up_start1
        This routine will initialize and start
        transmission.
*/
void rtc_up_start1()
{
        rad_tx_routine = rtc_rcv_up0;
        rad_timer = 0;
        if (rad_cfgx->base == MBA)
                rtc_tx_init(PORT0);
        else
                rtc_tx_init(port);
}


/*
        rtc_rcv_up0
        delays time for receiver not to be over runed
        called first in the 3 routine sequence for
        receiving data in slot.
*/
void rtc_rcv_up0()
{
    /* clear rts */
```

```
    ptt_off(port);
        if (rad_cfgx->base == MBA)
        {
                squelch[PORT0] = 0;
                squelch[PORT1] = 0;
        }

        else
                squelch[port] = 0;

        if ((rad_cfgx->remote_base[base] == REMOTE_ON) &&
                (rad_cfgx->remote_offset[base] != 0))
        {
                rad_timer_routine = rtc_rcv_up1;
                rad_timer = rad_cfgx->remote_offset[base];
        }
        else
        {
                rad_timer_routine = rtc_rcv_up2;
                rad_timer = PTT_DELAY-5;
        }
}
/*
        rtc_rcv_up1
        delays time for receiver not to be over runed
        called first in the 3 routine sequence for
        receiving data in slot.
*/
void rtc_rcv_up1()
{
        rad_timer_routine = rtc_rcv_up2;
        rad_timer = PTT_DELAY-5;
}

/*
        rtc_rcv_up2
        initialize the receiver to handle a slot
*/
void rtc_rcv_up2()
{

        if (rad_cfgx->base == MBA)
        {
                if ((rad_cfgx->mba[base] & PORTA) == PORTA)
                {
                        squelch_check[PORT0] = 1;
                        rtc_rx_init(PORT0);
                }
                if ((rad_cfgx->mba[base] & PORTB) == PORTB)
                {
                        squelch_check[PORT1] = 1;
                        rtc_rx_init(PORT1);
                }
        }
        else
        {
```

```
                squelch_check[port] = 1;
                rtc_rx_init(port);
        }

        rad_timer_routine = rtc_rcv_up3;
        if (scc_baud[base] == BAUD4800)
        {
                if (rtc_slots == 1)
                {
                        up_flag = 2;
                        rad_timer = 15;
                }
                else
                        rad_timer = UP_DELAY+4;
        }
        else
                rad_timer = UP9600_DELAY+5;
}

/*
        rtc_rcv_up3

        checks if a valid frame was received.
*/
WORD last;
void rtc_rcv_up3()
{
        last = 0xFF;
        /* turn off receiver */
        if (rad_cfgx->base == MBA)
        {
                if ((rad_cfgx->mba[base] & PORTA) == PORTA)
                {
                        rad_stop_scc(PORT0);
                        squelch_check[PORT0] = 0;
                        check_rcv_up(PORT0);
                }
                if ((rad_cfgx->mba[base] & PORTB) == PORTB)
                {
                        rad_stop_scc(PORT1);
                        squelch_check[PORT1] = 0;
                        check_rcv_up(PORT1);
                }

        }
        else
        {
                squelch_check[port] = 0;
                rad_stop_scc(port);
                check_rcv_up(port);
        }

   /* check if all slots have been checked */
   cslot++;
   if (cslot < rtc_slots)
          rtc_rcv_up1();
```

```
    else
            post_mailbox(0xFFFF);
}

check_rcv_up(port)
WORD port;
{
        if (rtc_rxx[port] == 1)
        {
           if (last != rtc_rx_buff[port]->address)
           {
                   last = rtc_rx_buff[port]->address;
                   up_q[up_q_count] = last & 0x7F;
                   up_q_count++;
           }
    }
    /* if squelch was detected and no valid data then
          indicate error
    */
    else if (squelch[port]==0xFFFF)
    {
           collisions++;
           channel_collisions[base]++;
           glob_collisions++;
    }
}
```

```
/*****************************************************************
****

Copyright (C)1991, Norand Corporation.  All Rights Reserved.


Revision History:

** $Log:   N:/3250/rtc/src/scc.c_v  $
 *
 *    Rev 1.6   27 Feb 1992 16:05:18   SOJKA
 * Dual Host Support Release and RS-422 Bug
 *
 *    Rev 1.5   18 Feb 1992 10:46:56   SOJKA
 * UIF cosmetic fixes V0.23
 *
 *    Rev 1.4   31 Dec 1991 11:22:06   sojka
 * Complied with Library V0.21.
 *
 *    Rev 1.3   31 Dec 1991 10:39:58   sojka
 * Complied with Library V0.21.
 *
 *    Rev 1.2   20 Nov 1991 18:36:50   sojka
 * Added Copyright Information and PVCS Log keyword.

*****************************************************************
****/



#include "main.h"
#include <configer.h>
#include <68302.h>
#include <int.h>
#include <interface.h>

#define PORT_NUM 2

scc_control_ptr scc_conx[PORT_NUM]     = {(scc_control_ptr)SCC3_CONTR
OL,
                                          (scc_control_ptr)SCC2_CONTR
OL};
scc_parm_ptr    scc_parmx[PORT_NUM]    = {(scc_parm_ptr)SCC3_PARM_RAM
,
                                          (scc_parm_ptr)SCC2_PARM_RAM
};

WORD scc_int_bit[PORT_NUM]             = {INT_SCC3_BIT,INT_SCC2_BIT};
LONG scc_vec[PORT_NUM]                 = {INT_SCC3,INT_SCC2};
WORD pin_mask[PORT_NUM]                  = {0x00FF,0xFF00};
WORD pin_ded[PORT_NUM]                 = (
                                          PORTA08+PORTA09+PORTA10+PORTA1
1+PORTA12,
                                          PORTA00+PORTA01+PORTA02+
```

```
    PORTA03+PORTA04+PORTA05+PORTA06+PORTA07,
                                        };

WORD cp_port[PORT_NUM]                          = { 0x04,0x02 };

WORD rts_onx[PORT_NUM]                            =  { RTS3, RTS2 };
WORD dtr_onx[PORT_NUM]                            =  { DTR3, DTR2 };

WORD alt1_onx[PORT_NUM]                           =  { ALT31, ALT21 };
WORD alt2_onx[PORT_NUM]                           =  { ALT32, ALT22 };

#define SCC_TX_ON    (PARITY_DISABLE+UART_MODE_NORMAL+STOP_BITS_1+DIA
G_SOFT+ENT+ENR+MODE_UART+BITS_8+UART_RTS)
/*
#define SCC_RX_ON    (PARITY_DISABLE+UART_MODE_NORMAL+STOP_BITS_1+DIA
G_SOFT+ENR+MODE_UART+BITS_8+UART_RTS)
*/
#define SCC_RX_ON    (SCC_TX_ON & (~UART_RTS))

int_struct_amx scc_int[PORT_NUM];

void (*rad_tx_routine)();

extern WORD rad_timer;
extern void (*rad_timer_routine)();
extern WORD up_flag;

#define TX_STATE0               0
#define TX_STATE1                       1
#define RX_STATE0                       2
#define RX_STATE1                       3
#define RX_STATE2                       4
#define RX_STATE3                       5
#define RX_STATE4                       6
#define NULL_STATE                      7


BYTE rx_buffer[PORT_NUM];
BYTE rad_tx_dummy[] = { 0x0f,0x0f};
#define SET_RX_TIMER(x,y) if (up_flag != 1) rad_timer = y

extern WORD crc_table[];

#define CALC_RX_CRC(x)  index = ((rtc_rx_crc[x] ^ (y & 0xff)) & 0x00
ff); \
                rtc_rx_crc[x] = (((rtc_rx_crc[x] >> 8) & 0x00ff) ^ c
rc_table[index]);


/*
    scc_interrupt
    handles the scc interrupt processing
*/
void scc_interrupt0()
{
        WORD index;
```

```
        BYTE y;
        BYTE mask;

        mask = 0xFF;
        y = rx_buffer[PORT0];

    ISR_PTR = scc_int_bit[PORT0];
    IPR_PTR = scc_int_bit[PORT0];
    scc_conx[PORT0]->scce     = 0xFF;
    switch (rad_state[PORT0])
    {
                /* send dummy frame */
                case TX_STATE0:
                        rad_state[PORT0] = TX_STATE1;

                        /* set scc to transmit */
                        scc_conx[PORT0]->scce     = 0xFF;
                        scc_conx[PORT0]->sccm     = UART_EVENT_TX;

                        /* set SCC buffer for proper transmit */
                        scc_parmx[PORT0]->txbd[0].buffer = &rad_tx_d
ummy[0];
                        scc_parmx[PORT0]->txbd[0].length = 2;
                        scc_parmx[PORT0]->txbd[0].flags = UART_TXBD_
R +

                  UART_TXBD_X +

                  UART_TXBD_W +

                  UART_TXBD_I;
                        break;

        /* end of transmit */
        case TX_STATE1:
            scc_conx[PORT0]->sccm     = 0x00;
            (rad_tx_routine)();
            break;

        /* wait for STX */
        case RX_STATE0:
            /* if stx save it and go to next state */
            if (y == STX)
            {
                rad_state[PORT0] = RX_STATE1;
                                rtc_rx_crc[PORT0] = 0;
                                goto set_timer0;
            }
                        else
                                scc_parmx[PORT0]->rxbd[0].flags = UA
RT_RXBD_E +

                          UART_RXBD_X +
                                                UART_RXBD_W +
                                                UART_RXBD_I;
            break;
```

```
        /* RX_STATE1
            -- check for STX
                if STX found, abort recieve
            -- check for ETX
                if ETX found, go to state 3
            -- check for DLE
                if DLE found, go to state 2
            -- otherwise
                place data into data field buffer.
        */
        case RX_STATE1:
            if (y == STX)
                                    goto error0;
            else if (y == ETX)
                                    {
                rad_state[PORT0] = RX_STATE3;
                                    goto calc_crc0;
                                    }
            else if (y == DLE)
                                    {
                rad_state[PORT0] = RX_STATE2;
                                    goto calc_crc0;
                                    }
            else
                                    goto save_data0;
            break;

        /* RX_STATE2
            -- otherwise
                place data into data field buffer with high bit clea
red.
                goto state 1;
        */
        case RX_STATE2:
           rad_state[PORT0] = RX_STATE1;
                  mask = 0x7f;
save_data0:
            if (rtc_rx_length[PORT0] >= FRAME_SIZE)
                                    goto error0;
            else
            {


                                    rtc_rx_length[PORT0]++;
                                    *(rtc_rxptr[PORT0])++ = y & mask;
calc_crc0:
                                    CALC_RX_CRC(PORT0);
set_timer0:
                                    SET_RX_TIMER(PORT0,4);
                                    scc_parmx[PORT0]->rxbd[0].flags = UA
RT_RXBD_E + UART_RXBD_X +

                                UART_RXBD_W + UART_RXBD_I;
                                    return;
            }
```

```
        /* receive first crc */
        case RX_STATE3:
            if ((rtc_rx_crc[PORT0] & 0xFF) == y)
                    {
                rad_state[PORT0] = RX_STATE4;
                                goto set_timer0;
                    }
            else
                                goto error0;
            break;

        /* receive second crc */
        case RX_STATE4:
            if ((((rtc_rx_crc[PORT0]) >> 8) & 0xFF) == y)
            {
                rtc_rxx[PORT0] = 1;
                rad_state[PORT0] = NULL_STATE;
                if (up_flag == 0)
                {
                    post_mailbox(0xFFFF);
                    rad_timer = 0;
                }
                                else if (up_flag == 2)
                                        rad_timer = 1;
            }
            else
                                goto error0;
            break;

        case NULL_STATE:
            scc_conx[PORT0]->sccm      = 0x00;
            break;
    }
        return;

error0:
        rad_state[PORT0] = NULL_STATE;
        SET_RX_TIMER(PORT0,1);
        return;

}
/*
    scc_interrupt
    handles the scc interrupt processing
*/
void scc_interrupt1()
{
        WORD index;
        BYTE y;
        BYTE mask;

        mask = 0xFF;
        y = rx_buffer[PORT1];

    ISR_PTR = scc_int_bit[PORT1];
    IPR_PTR = scc_int_bit[PORT1];
```

```
        scc_conx[PORT1]->scce      = 0xFF;
        switch (rad_state[PORT1])
        {
                    /* send dummy frame */
                    case TX_STATE0:
                            rad_state[PORT1] = TX_STATE1;

                            /* set scc to transmit */
                            scc_conx[PORT1]->scce     = 0xFF;
                            scc_conx[PORT1]->sccm     = UART_EVENT_TX;

                            /* set SCC buffer for proper transmit */
                            scc_parmx[PORT1]->txbd[0].buffer = &rad_tx_d
ummy[0];

                            scc_parmx[PORT1]->txbd[0].length = 2;
                            scc_parmx[PORT1]->txbd[0].flags = UART_TXBD_
R +

                             UART_TXBD_X +

                             UART_TXBD_W +

                             UART_TXBD_I;
                                    break;
                /* end of transmit */
                case TX_STATE1:
                    scc_conx[PORT1]->sccm      = 0x00;
                    (rad_tx_routine)();
                    break;

                /* wait for STX */
                case RX_STATE0:
                    /* if stx save it and go to next state */
                    if (y == STX)
                    {
                        rad_state[PORT1] = RX_STATE1;
                                        rtc_rx_crc[PORT1] = 0;
                                        goto set_timer1;
                    }
                                else
                                        scc_parmx[PORT1]->rxbd[0].flags = UA
RT_RXBD_E +

                                  UART_RXBD_X +

                                  UART_RXBD_W +

                                  UART_RXBD_I;
                    break;

                /* RX_STATE1
                    -- check for STX
                        if STX found, abort recieve
                    -- check for ETX
                        if ETX found, go to state 3
                    -- check for DLE
```

```
                if DLE found, go to state 2
            -- otherwise
                place data into data field buffer.
        */
        case RX_STATE1:
            if (y == STX)
                                goto error1;
            else if (y == ETX)
                        {
                rad_state[PORT1] = RX_STATE3;
                                goto calc_crc1;
                        }
            else if (y == DLE)
                        {
                rad_state[PORT1] = RX_STATE2;
                                goto calc_crc1;
                        }
            else
                                goto save_data1;
            break;

        /* RX_STATE2
            -- otherwise
                place data into data field buffer with high bit clea
red.
                goto state 1;
        */
        case RX_STATE2:
                rad_state[PORT1] = RX_STATE1;
                                mask = 0x7f;
save_data1:
            if (rtc_rx_length[PORT1] >= FRAME_SIZE)
                                goto error1;
            else
            {
                                rtc_rx_length[PORT1]++;
                                *(rtc_rxptr[PORT1])++ = y & mask;
calc_crc1:
                                CALC_RX_CRC(PORT1);
set_timer1:
                                SET_RX_TIMER(PORT1,4);
                                scc_parmx[PORT1]->rxbd[0].flags = UA
RT_RXBD_E + UART_RXBD_X +

                                UART_RXBD_W + UART_RXBD_I;
                                return;
            }

        /* receive first crc */
        case RX_STATE3:
            if ((rtc_rx_crc[PORT1] & 0xFF) == y)
                        {
                rad_state[PORT1] = RX_STATE4;
                                goto set_timer1;
                        }
            else
```

```
                                goto error1;
            break;

        /* receive second crc */
        case RX_STATE4:
            if ((((rtc_rx_crc[PORT1]) >> 8) & 0xFF) == y)
            {
                rtc_rxx[PORT1] = 1;
                rad_state[PORT1] = NULL_STATE;
                if (up_flag == 0)
                {
                    post_mailbox(0xFFFF);
                    rad_timer = 0;
                }
            }
            else
                                goto error1;
            break;

        case NULL_STATE:
            scc_conx[PORT1]->sccm      = 0x00;
            break;
    }
        return;

error1:
        rad_state[PORT1] = NULL_STATE;
        SET_RX_TIMER(PORT1,1);
        return;

}


void scc_setup()
{
        if (rad_cfgx->port == COMM_PORT2)
        {
                scc_conx[PORT0] = scc_conx[PORT1];
                scc_parmx[PORT0] = scc_parmx[PORT1];
                scc_int_bit[PORT0] = scc_int_bit[PORT1];
                scc_vec[PORT0] = scc_vec[PORT1];
                pin_mask[PORT0] = pin_mask[PORT1];
                pin_ded[PORT0] = pin_ded[PORT1];
                rts_onx[PORT0] = rts_onx[PORT1];
                dtr_onx[PORT0] = dtr_onx[PORT1];
                alt1_onx[PORT0] = alt1_onx[PORT1];
                alt2_onx[PORT0] = alt2_onx[PORT1];

        }

        if (rad_cfgx->base == NORMAL_BASE)
        {
                port = PORT0;
                base = PORT0;
                scc_init(PORT0);
        }
```

```
        else if (rad_cfgx->base == DUAL_BASE)
        {
                port = PORT0;
                base = PORT0;
                scc_init(PORT0);
                scc_init(PORT1);
        }
        /* MBA support */
        else if (rad_cfgx->base == MBA)
        {
                port = PORT0;
                base = PORT0;
                scc_init(PORT0);
                scc_init(PORT1);
        }
        /* MBA SEQ support */
        else if (rad_cfgx->base == MBA_SEQ)
        {
                port = PORT0;
                base = PORT0;
                scc_init(PORT0);
                rad_cfgx->base = MBA;
        }
}


void scc_init(port)
WORD port;
{
    WORD x;

    /* set initial interrupts */
        if (port == PORT0)
                set_interrupt_amx(scc_vec[port],(LONG)scc_interrupt0
,&scc_int[port]);
        else
                set_interrupt_amx(scc_vec[port],(LONG)scc_interrupt1
,&scc_int[port]);

    ajdi();
    /* initialize pin ports */
    x = PACNT_PTR;
        /* save the other port functions */
    x &= pin_mask[port];
    /* make data and clock lines dedicated */
    x |= pin_ded[port];
    PACNT_PTR = x;
    /* make ptt and hs pins output lines */

        /* turn off rts/dtr */
        ptt_off(port);
        dtr_off(port);
        alt_off();

    /* initialize the scc control strucuture */
    scc_conx[port]->scon     = BAUD4800;
```

```
                default:
        case MBA:
                        if ((rad_cfgx->mba[base] & PORTA) == PORTA)
                                rad_stop_scc(PORT0);
                        if ((rad_cfgx->mba[base] & PORTB) == PORTB)
                                rad_stop_scc(PORT1);
            if (!((rad_cfgx->speed[base] == XBAUD_DUAL) &&
                (scc_baud[base] == BAUD9600)))
            {
                                i = base;
                base = (base + 1) % 4;
                while ((!(rad_cfgx->mba[base])) && (base != i))
                    base = (base + 1) % 4;
            }
            port = 0;
            break;
    }

        if (rad_cfgx->remote_base[base] == REMOTE_ON)
                lcd_config(LCD_REMOTE);
        else
                lcd_config(rad_cfgx->base);

    switch (rad_cfgx->speed[base])
    {

                case XBAUD_4800:
                        lcd_speed(LCD_4800);
                        scc_baud[base] = BAUD4800;
                        dtr_off(port);
                        break;

        case XBAUD_DUAL:
                        if (rad_cfgx->remote_base[base] == REMOTE_ON
)
                        {
                                lcd_speed(LCD_4800);
                                scc_baud[base] = BAUD4800;
                                dtr_off(port);
                        }
                        else if (scc_baud[base] == BAUD4800)
                        {
                                lcd_speed(LCD_9600);
                                scc_baud[base] = BAUD9600;
                                dtr_on(port);
                        }
                        else
                        {
                                lcd_speed(LCD_4800);
                                scc_baud[base] = BAUD4800;
                                dtr_off(port);
                        }
                        break;

                case XBAUD_9600:
                        lcd_speed(LCD_9600);
```

```
                        scc_baud[base] = BAUD9600;
                        dtr_on(port);
                        break;

    }


    if (rad_cfgx->base == MBA)
        {
        set_mba(base);
                if ((rad_cfgx->mba[base] & PORTA) == PORTA)
                        scc_conx[PORT0]->scon = scc_baud[base];
                if ((rad_cfgx->mba[base] & PORTB) == PORTB)
                        scc_conx[PORT1]->scon = scc_baud[base];
        }
        else
                scc_conx[port]->scon = scc_baud[base];

        lcd_port(base+1);
}


void dtr_on(port)
WORD port;
{
        RTS_DTR_PTR &= (~dtr_onx[port]);
}

void dtr_off(port)
WORD port;
{
        RTS_DTR_PTR |= dtr_onx[port];
}

void ptt(port)
WORD port;
{
        RTS_DTR_PTR &= (~rts_onx[port]);
        while (*CP_PTR & CP_FLAG)
        {}
        *CP_PTR = CP_RESTART_TRANSMIT |
                        cp_port[port] |
                            CP_FLAG;

    scc_conx[port]->scm      = SCC_TX_ON;
}

void ptt_off(port)
WORD port;
{
        RTS_DTR_PTR |= rts_onx[port];
    scc_conx[port]->scm      = SCC_RX_ON;

}

void set_mba(base)
```

```
WORD base;
{
        WORD x,index;
        x = RTS_DTR_PTR;
        if (rad_cfgx->port == COMM_PORT2)
                index = 1;
        else
                index = 0;

        if (base & 0x01)
                x |= alt1_onx[index];
        else
                x &= (~alt1_onx[index]);
        if (base & 0x02)
                x |= alt2_onx[index];
        else
                x &= (~alt2_onx[index]);
        RTS_DTR_PTR = x;
}

void alt_off()
{
        WORD x,index;
        x = RTS_DTR_PTR;
        if (rad_cfgx->port == COMM_PORT2)
                index = 1;
        else
                index = 0;
        RTS_DTR_PTR |= (alt1_onx[index] | alt2_onx[index]);
}

WORD check_squelch(port)
WORD port;
{
        WORD i;

        if (rad_cfgx->signalling[port] == SIGNAL_RS422)
        {
                i = scc_conx[port]->sccs & SCCS_CTS;
                if (rad_cfgx->remote_base[base] == REMOTE_ON)
                        i ^= SCCS_CTS;
        }
        else
        {
                i = scc_conx[port]->sccs & SCCS_CD;
                if (rad_cfgx->remote_base[base] == REMOTE_ON)
                        i ^= SCCS_CD;
        }
    return i;
}

void rad_stop_scc(port)
WORD port;
{
    /* disable scc */
    scc_conx[port]->scce     = 0xFF;
```

```
    scc_conx[port]->sccm     = 0x00;

    /* set state to NULL state */
    rad_state[port]          = NULL_STATE;
    scc_parmx[port]->rxbd[0].flags = UART_RXBD_X +
                                     UART_RXBD_W +
                                     UART_RXBD_I;
    scc_parmx[port]->txbd[0].flags = UART_TXBD_X +
                                     UART_TXBD_W +
                                     UART_TXBD_I;
}


void rtc_tx_init(port)
WORD port;
{
    /* set state to tx state */
    rad_state[port] = TX_STATE0;

    /* set scc to transmit */
    scc_conx[port]->scce     = 0xFF;
    scc_conx[port]->sccm     = UART_EVENT_TX;

    /* set SCC buffer for proper transmit */
    scc_parmx[port]->txbd[0].buffer = &rad_tx_buffer[0];
    scc_parmx[port]->txbd[0].length = rad_tx_length+2;
    scc_parmx[port]->txbd[0].flags = UART_TXBD_R +
                                     UART_TXBD_X +
                                     UART_TXBD_W +
                                     UART_TXBD_I;
}

void rtc_rx_init(port)
WORD port;
{
    /* set state and length for receive */
    rad_state[port]     = RX_STATE0;
    rtc_rx_length[port] = 0;
        rtc_rxptr[port] = &(rtc_rx_buff[port]->address);
        rtc_rxx[port] = 0;

    /* set scc for receive */
    scc_conx[port]->scce     = 0xFF;
    scc_conx[port]->sccm     = UART_EVENT_RX;

    scc_conx[port]->scce     = 0xFF;
    scc_parmx[port]->rxbd[0].flags = UART_RXBD_E +
                                     UART_RXBD_X +
                                     UART_RXBD_W +
                                     UART_RXBD_I;
}


void scc_send_one(port)
{
    /* set state to tx state */
```

```
    rad_state[port] = NULL_STATE;

    /* set scc to transmit */
    scc_conx[port]->scce     = 0xFF;
    scc_conx[port]->sccm     = UART_EVENT_TX;

    /* set SCC buffer for proper transmit */
    scc_parmx[port]->txbd[0].buffer = &rad_tx_dummy[0];
    scc_parmx[port]->txbd[0].length = 1;
    scc_parmx[port]->txbd[0].flags = UART_TXBD_R +
                                     UART_TXBD_X +
                                     UART_TXBD_W;
}
```

```
#include <config.h>
#include <queue.h>
#include <buffer.h>
#include <priority.h>
#include <rtc.h>
#include "rad_cfg.h"
#include "rad.h"
#include "radext.h"
#include "rtc.pro"
```

```
/*
        rf.h -- header file for the rtc master protocol for the rt32
10
        author -- Marvin Sojka 1/19/88
*/
#ifndef RAD_H
#ifndef NULL
#define NULL ((void *)0)
#endif

#define MAX_TERMS        128

typedef struct RCB
{
        WORD free;                                              /* 1
- inuse */
        WORD address;                                   /* address f
or RCB */
        WORD vr;                                                /* a
ck0 or ack1 for receive */
        WORD vs;                                                /* a
ck0 or ack1 for send */
        WORD state;                                             /* c
urrent state of RCB */
        buff_ptr txq;                                           /* t
x queue */
        buff_ptr rxq;                                           /* r
x queue */
        WORD rx_length;                                 /* rx_length
 */
        WORD inact_timer;                               /* inact tim
er value */
        WORD long_timer;
        WORD type;                                              /* t
erminal type */
        WORD retrys;                                    /* retry cou
nter */
        WORD retry_max;
        WORD last_base;
        WORD last_speed;
        LONG time;
} rcb,*rcb_ptr;

#define RCB_FREE 0
#define RCB_INUSE 1

/* RTC event units */
#define IDLE0_EVENT     0x00
#define IDLE1_EVENT     0x01
#define IDLE2_EVENT     0x02
#define IDLE3_EVENT     0x03
#define IDLE4_EVENT     0x04
#define TIMEOUT_EVENT   0x05
#define RESET_EVENT     0x06
#define TEST_EVENT              0x07
#define VERSION_EVENT   0x08
```

```
#define ACK0_EVENT          0x09
#define ACK1_EVENT      0x0A
#define ACK2_EVENT      0x0B
#define ACKN0_EVENT     0x0C
#define ACKN1_EVENT     0x0D
#define ACKN2_EVENT     0x0E
#define D0_EVENT                0x0F
#define D1_EVENT                0x10
#define D2_EVENT                0x11
#define TERROR_EVENT    0x12
#define IDLE5_EVENT             0x13
#define TX_TIMEOUT_EVENT 0x14

/* states of the rts state machines */

#define RESET_STATE 0x00
#define NORM_STATE 0x01
#define RSP_STATE 0x02
#define TEST_STATE 0x03
#define VERSION_STATE 0x04
#define WAIT_STATE 0x05
#define ECHO_STATE 0x06

/* state machine return codes */
#define NOT_DONE      0x00
#define DONE          0x01
#define RESET_CODE        0x02
#define CONNECT_CODE  0x03


/* remote base equates */
#define REMOTE_OFF 0
#define REMOTE_ON 1

/* MSTAT EQUATES */
#define MSTAT_NOT_ACTIVE                1
#define MSTAT_ACTIVE                    2
#define MSTAT_MESSAGE                   3
#define MSTAT_INACTIVE_MESSAGE  4
#define MSTAT_INACTIVE                  5

/*
        Default Timing parameters
*/
#define SLOW_PTT                45
 /* GE Radio PTT Time */
#define FAST_PTT                15
 /* Current PTT Time */
#define PTT_DELAY               15
 /* Default PTT Time */
#define UP_DELAY                25
 /* 4800 baud UP Window */
#define UP9600_DELAY    10
600 baud UP Window */                                   /* 9
#define GAP_DELAY               4
 /* GAP Timeout Value */
```

```
#define BLOCK_DELAY             GAP_DELAY*2                     /* I
nter Block Timeout Value */
#define MAX_NULL_UP             5


#define FRAME_SIZE                      135

#define TEN_SECONDS                     1000

#define RTC_MIN_SLOTS           1
#define RTC_MAX_SLOTS           11
#define RTC_SLOT_COUNT          3

#define RTC_OVERHEAD            2


/* rad lcd tx status equates */
#define LCD_TX_UP                               0
#define LCD_TX_POLL                             1
#define LCD_TX_DATA                             2
#define LCD_TX_TEST                             3
#define LCD_TX_VERSION                  4
#define LCD_TX_RESET                    5
#define LCD_TX_INTERFERENCE             6
#define LCD_TX_CALLSIGN                 7

/* rad lcd rx equates */
#define LCD_RX_GOOD                             0
#define LCD_RX_TIMEOUT                  1
#define LCD_RX_CRC_ERROR                2

/* rad lcd status equates                */
#define RAD_GOOD                                0
#define RAD_RADIO                               1
#define RAD_HOST                                2

/* rad lcd rf speed equates */
#define LCD_4800                                0
#define LCD_9600                                1

/* rad config equates    */
#define LCD_RTC                                 0
#define LCD_DUAL                                1
#define LCD_MBA                                 2
#define LCD_MBA_SEQ                             3
#define LCD_REMOTE                              4

#define CRICKET_SHIT                    71

#endif
```

```
#ifndef RAD_CFG_H
#define RAD_CFG_H

typedef struct
{
        queue_ptr rad_in_q;
        queue_ptr rad_out_q;
        WORD      port;
        WORD      sna;
        WORD      signalling[2];
        BYTE      fcc_callsign[16];
        WORD      base;
        WORD      mba[4];
        WORD      speed[4];
        WORD      remote_base[4];
        WORD      remote_offset[4];
} rad_cfg,*rad_cfg_ptr;

#define COMM_PORT2              2
#endif
```

```
     SUBTTL SCC_EQUATES
     PAGE

;        interrupt controller constants
;        for non specific end of interrupt
INTERNAL_PORT_BASE  equ 0FF00H
EOI_REG             equ INTERNAL_PORT_BASE+22H
EOI                 equ 8000H                    ;non-specific end-of
-interrupt

VRTX            equ     20H
UI_ENTER        equ     16H
UI_EXIT         equ     11H
SC_QINQUIRY equ 2AH
SC_POST         equ     08H

LCD_Q           equ     25

EOI_8259_REG        equ 0fa00h
EOI_8259            equ 020h                     ;non-specific end-of
-interrupt


;
;
;    SCC_EQUATES
;
;register 0 ("command register")
IUS                 equ 038H            ;reset highest Interrupt Und
er Service
RESET_ERROR         equ 030H
RESET_STATUS        equ 010H
RESET_TX_CRC        equ 080H
RESET_RX_CRC        equ 040H
RESET_RX            equ 070H
RESET_TX_UNDERRUN   equ 0C0H
RESET_TX_PENDING    equ 028H
;
ABORT_ON_UNDERRUN   equ 0A4H
FCS_ON_UNDERRUN     equ 0A0H

;register 1 (interrupt control)
TX_INTS             equ 00000010b   ;enable tx ints
EXT_TX_INTS         equ 00000011b   ;enable tx, ext stat ints
EXT_INTS            equ 00000001b   ;enable ext stat ints
RX_INTS             equ 00010100b   ;enable rx ints
NO_INTS             equ 00000000b   ;set  dma bits

;register 5 (tx control)
TX_ON               equ 0AH
PTT                                 equ 0AH

;register 3 (rx control)
RX_ON               equ 01H

;register 9
```

```
RESETA                          equ 089H
RESETB                          equ 049H

;register 10
CRCPRESET1                      equ 080H
NRZI                            equ 020H
NRZ                                     equ 000H

;register 11
CLOCK                           equ 078H
CLOCK1                          equ 008H
CLOCK2                          equ 052H

;register 14
SNRZI                           equ 0e0H
BRGEN                           equ 080H
DDPLL                           equ 060H
SEARCH                          equ 020H
BRENABLE                        equ 003H

;register 15 (external/status interrupt control)
TX_UND_INT          equ 040H        ;enable tx underrun interrupt
CD_INT              equ 008H        ;enable CD status interrupt
CTS_INT             equ 020H        ;enable CTS status interrupt
ABORT_INT           equ 080H        ;enable ABORT status interrupt


;read reg 0 meanings
DCD                 equ 08h
CTS                 equ 20h
ABORT               equ 80h
UNDERRUN            equ 40h
TX_EMPTY                        equ     04H

;read register 1 test masks
EOFRAME             equ 80h
CRCFRMERR           equ 40h
RXOVERRUN           equ 20h
PARERR              equ 10h

;protocol parameters
SDLC_MODE                       equ 020H
ASYNC_MODE                      equ 044H
SDLC_FLAG                       equ 07EH
;
MAX_IN                          equ 266
    SUBTTL MACROS
    PAGE
; read_reg
;   read from a specified scc register. the port address is optional
ly
;   specified, so that you don't have to reload dx unless you need t
o.
;   Also, reads from port 0 don't generate a spurious write to set u
p the
;   register
```

```
;
read_reg    macro   regnum,portnum ;19;39;54
ifnb    <portnum>   ;if a port address is specified
    mov dx,portnum                  ;16;  ;  ;16;
endif
if 0 ne regnum
    mov al,regnum                   ; 4;  ; 4;20;
    out dx,al                       ; 9;  ;13;29;
    nop                             ; 3;  ;16;32;
    nop                             ; 3;  ;19;35;
    nop                             ; 3;  ;22;38;
    nop                             ; 3;  ;16;32;
    nop                             ; 3;  ;19;35;
    nop                             ; 3;  ;22;38;
endif
    in  al,dx                       ; 8; 8;30;46;
    nop                             ; 3;12;33;49;
    nop                             ; 3;15;36;51;
    nop                             ; 3;19;39;54;
    nop                             ; 3;  ;16;32;
    nop                             ; 3;  ;19;35;
    nop                             ; 3;  ;22;38;
endm

; write_reg
;    opposite of read_read. same notes as above apply.
;
write_reg   macro   regnum,val,portnum ;20;40;52;
ifnb    <portnum>   ;if a port address is specified
    mov dx,portnum                      ;12;  ;  ;12;
endif
if 0 ne regnum
    mov al,regnum                       ; 4;  ; 4;16;
    out dx,al                           ; 7;  ;11;23;
    nop                                 ; 3;  ;14;26;
    nop                                 ; 3;  ;17;29;
    nop                                 ; 3;  ;20;32;
    nop                             ; 3;  ;16;32;
    nop                             ; 3;  ;19;35;
    nop                             ; 3;  ;22;38;
endif
    mov al,val                          ; 4; 4;24;36;
    out dx,al                           ; 7;11;31;43;
    nop                                 ; 3;14;34;46;
    nop                                 ; 3;17;37;49;
    nop                                 ; 3;20;40;52;
    nop                             ; 3;  ;16;32;
    nop                             ; 3;  ;19;35;
    nop                             ; 3;  ;22;38;
endm
; write_reg
;    opposite of read_read. same notes as above apply.
;
write_regx  macro   regnum,val,portnum ;20;40;52;
ifnb    <portnum>   ;if a port address is specified
    mov dx,portnum                      ;12;  ;  ;12;
```

```
 endif
 if 0 ne regnum
      mov al,regnum
      out dx,al                                  ; 4;   ; 4;16;
      nop                                        ; 7;   ;11;23;
      nop                                        ; 3;   ;14;26;
      nop                                        ; 3;   ;17;29;
      nop                                        ; 3;   ;20;32;
      nop                                  ; 3;   ;16;32;
      nop                                  ; 3;   ;19;35;
      nop                                  ; 3;   ;22;38;
 endif
      mov ax,val
      out dx,al                                  ; 4; 4;24;36;
      nop                                        ; 7;11;31;43;
      nop                                        ; 3;14;34;46;
      nop                                        ; 3;17;37;49;
      nop                                        ; 3;20;40;52;
      nop                                  ; 3;   ;16;32;
      nop                                  ; 3;   ;19;35;
      nop                                  ; 3;   ;22;38;
 endm

      PAGE
 ;
 ;
 ; signal_eoi                                ;rel 062287
 ;    resets scc interrupt and interrupt controllers
 ;
 signal_eoi  macro                                      ;126;

 ;    reset the status and error state of the scc
      write_reg   0,IUS,scc_command_port[si] ;32;32;
      write_reg   0,RESET_ERROR                         ;20;52;
 ;
 ;    reset the 80188 internal interrupt controller
      mov     dx,EOI_REG                                ;16;103;
      mov     ax,EOI                                    ;12;115;
      out     dx,ax                                     ;11;126;

 endm

 ;*****************************************************************************
 **********
 ;    THESE EQUATES THE 'ASCII' CONTROL CHARACTERS
 ;*****************************************************************************
 **********
 STX                    EQU     02H          ;START OF TEXT
 ETX                    EQU     03H          ;END OF TEXT
 DLE                                             EQU                  10H
                  ;DATA-LINK ESCAPE
 ;
 ; LCD EQUATES
 ;
 waity macro
         LOCAL loop
         mov dx,LATCH
         mov al,FAKE_READ
```

```
        out dx,al
        mov dx,LCD_INIT
loop:
        in al,dx
        test al,BUSY_FLAG
        jnz     loop
;
;       call fakewrite
        mov dx,LATCH
        mov al,FAKE_WRITE
        out dx,al
endm



waitx macro
        LOCAL loop
        mov dx,LATCH
        mov al,FAKE_READ
        out dx,al
        mov dx,LCD_INIT
loop:
        in al,dx
        test al,BUSY_FLAG
        jnz     loop
;
;       call FAKE_READ
endm


        SUBTTL  Equates and macros
LCD_BASE                    equ     200H
LCD_INIT                    equ     LCD_BASE+0
LCD_DATA                    equ     LCD_BASE+2
;other equates
LCD_INIT0                   equ     00111000B
LCD_INIT1                   equ     00000110B
CURSOR_OFF                  equ     00001100B
DISPLAY_ON                  equ     00001110B
BUSY_FLAG                   equ     10000000B
CLEAR                       equ     00000001B
;
LATCH                       equ     0180h
FAKE_READ                   equ     2h
FAKE_WRITE                  equ     00h
ADDRESS_MASK      equ     07FH
SET_DD_RAM_ADDRESS equ    080H
;
SCREEN_SIZE                 equ     20H
LINE_SIZE                   equ     10H
SECOND_LINE                 equ     40H
ADJUSTMENT                  equ     30H
```

```
SUBTTL SEGMENT DEFINITIONS, DATA STRUCTURES
;DATA  segment public 'DATA'
    extrn scc_command_port:word
    extrn scc_data_port:word
        extrn scc_base_pos:word
        extrn scc_vector:word
        extrn scc_baud:word
        extrn scc_reset:word
        extrn port:word
        extrn mailbox:word
        extrn base:word
        extrn base_mul:word
        extrn rtc_rxx:word
;
    extrn rtc_tx_state:word
    extrn rtc_tx:word
    extrn rtc_tx_length:word
        extrn rtc_address:byte
        extrn rtc_command:byte
        extrn rtc_tx_crc:byte
;
    extrn rtc_rx_state:word
    extrn rtc_rxptr:word
    extrn rtc_rxptr1:word
    extrn rtc_rx_length:word
        extrn rtc_rx_command:word
        extrn rtc_rx_address:word
        extrn rtc_rx_crc:word
;
;
        extrn rad_timer_routine:word
        extrn rtc_block_delay:word
        extrn rtc_gap_delay:word
;
    extrn rad_timer:word
;
        extrn squelch:word
        extrn squelch_check:word
        extrn up_flag:word
;
        extrn native_length:word
        extrn native_error:byte
;
        extrn mode_byte:byte
        extrn rx_byte:byte
        extrn tx_byte:byte
;
        extrn sna:word
;
        extrn crc_table:word
```

```
/*
        rad.c
        author Marvin Sojka

        purpose:
                Declarations for global identies used in the RTC pro
tocol
*/
extern rad_cfg_ptr rad_cfgx;

extern void (*rad_timer_routine)();
extern void (*rad_tx_routine)();
extern rcb rcbs[];

extern WORD up_q[];
extern WORD up_q_count;
extern WORD up_flag;
extern WORD squelch[];
extern WORD squelch_check[];
extern WORD rtc_slots;

extern rcb_ptr  crcb;
extern buff_ptr cbuff;

extern buff_ptr rtc_rx_buff[];
extern BYTE *rtc_rxptr[];
extern BYTE *rtc_rxptr1[];

extern BYTE rad_tx_buffer[];
extern WORD rad_tx_length;
extern WORD rtc_rx_length[];
extern WORD rtc_rx_crc[];
extern WORD rad_state[];

extern WORD rad_timer;
extern WORD rad_inact_timeout;

extern WORD rad_callsign_timer;

extern WORD sna;
extern WORD snax;

extern WORD port;
extern WORD base;
extern WORD rtc_rxx[];

extern WORD up_inactive_count[];
extern WORD fup_inactive_count[];
extern WORD slot[];
extern WORD fslot[];
extern WORD scc_baud[];

/* stats */
extern LONG     term_out_blocks[];
extern LONG     term_in_blocks[];
extern LONG     term_out_timeouts[];
```

```
extern LONG     term_in_timeouts[];

extern LONG     glob_out_blocks;
extern LONG     glob_in_blocks;
extern LONG     glob_out_timeouts;
extern LONG     glob_in_timeouts;
extern LONG     glob_interference;
extern LONG glob_collisions;

extern LONG     channel_out_blocks[];
extern LONG     channel_in_blocks[];
extern LONG     channel_out_timeouts[];
extern LONG     channel_in_timeouts[];
extern LONG     channel_interference[];
extern LONG channel_collisions[];
```

```
/* rtc_pri.c */
unsigned short rtc_prifsm(int event);
unsigned short rtc_reset_state(unsigned short event);
unsigned short rtc_norm_state(unsigned short event);
unsigned short rtc_rsp_state(unsigned short event);
unsigned short rtc_test_state(unsigned short event);
unsigned short rtc_version_state(unsigned short event);
/* rtc_init.c */
void rtc_init(void);
/* rtc_ind.c */
int rtc_reset_indication(void);
int rtc_data_indication(buff_ptr nbuff);
/* rtc_up.c */
void rtc_up(void);
int rtc_send_up(unsigned short slots);
void rtc_up_squelch(void);
void rtc_up_start0(void);
void rtc_up_start1(void);
void rtc_rcv_up0(void);
void rtc_rcv_up1(void);
void rtc_rcv_up2(void);
void rtc_rcv_up3(void);
int check_rcv_up(unsigned short port);
/* rtc_idle.c */
int rtc_idle(void);
/* rtc_rx.c */
unsigned short rtc_rx(unsigned short who);
unsigned short next_rx_buffer(void);
/* rtc_acti.c */
unsigned short rtc_rcv_data(unsigned char command,unsigned short len
gth);
unsigned short rtc_accept_data(void);
void rtc_accept_special(void);
void rtc_reject_echo(rcb_ptr qrcb);
int rtc_rcv_test(unsigned short length);
int rtc_rcv_version(unsigned short length);
void rtc_reset(rcb_ptr qrcb);
void rtc_done(int type);
/* rtc_send.c */
int rtc_send_poll(void);
int rtc_send_npoll(void);
int rtc_send_reset(void);
int rtc_send_data(void);
int rtc_send_version(void);
int rtc_send_test(void);
int rtc_send(unsigned char address,unsigned char command,unsigned ch
ar *data,unsigned short length);
void rtc_tx_start0(void);
/* rtc_main.c */
void rtc_main(void);
void rtc_rx_start0(void);
void rtc_rx_start1(void);
unsigned short rtc_receive(void);
void rad_timeout(void);
/* rad_main.c */
void rad_main(void);
```

```
/* rad_inf.c */
void send_main(buff_ptr x);
buff_ptr wait_main(unsigned int timeout);
buff_ptr check_main(void);
void post_main(buff_ptr x,queue_ptr que);
int post_mailbox(unsigned int x);
unsigned int pend_mailbox(unsigned int x);
void mailbox_init(void);
/* rad_in.c */
void rad_in(void);
void rf_halt_in(void);
void rf_data_in(buff_ptr qbuff);
void rf_reboot_in(buff_ptr qbuff);
void rf_enable_in(buff_ptr qbuff);
void rf_mstat_in(buff_ptr qbuff);
void rf_stat_in(buff_ptr qbuff);
void rf_stat_reset_in(buff_ptr qbuff);
void rf_type_in(buff_ptr qbuff);
int convert_long_to_bytes(unsigned char *x,unsigned int y);
/* rad_rcb.c */
rcb_ptr rcb_alloc(unsigned short address);
rcb_ptr rcb_find(unsigned short address);
void rcb_free(rcb_ptr xrcb);
void rcb_init(void);
/* rad_time.c */
void timer_interrupt(void);
void rad_inact(AMXID timerid,unsigned int a5);
void rad_timer_init(void);
/* rad_lcd.c */
void lcd_startup(void);
void lcd_update(void);
void lcd_addr(unsigned short addr);
void lcd_tx_status(unsigned short status);
void lcd_rx_status(unsigned short status);
void lcd_heartbeats(unsigned short heartbeats);
void lcd_port(unsigned short port);
void lcd_speed(unsigned short speed);
void lcd_config(unsigned short config);
void lcd_sys_status(unsigned short status);
/* rad.c */
/* rad_stat.c */
void stats_init(void);
void clear_term_stats(unsigned short address);
/* \lib/config.h */
void main(rad_cfg_ptr x);
/* \lib/config.h */
unsigned short calc_crc(unsigned char *buff,unsigned short len);
unsigned short calc_rx_crc(unsigned short crc,unsigned char new);
/* \lib/config.h */
void scc_interrupt0(void);
void scc_interrupt1(void);
void scc_setup(void);
void scc_init(unsigned short port);
void rtc_switch(void);
void dtr_on(unsigned short port);
void dtr_off(unsigned short port);
```

```
void ptt(unsigned short port);
void ptt_off(unsigned short port);
void set_mba(unsigned short base);
void alt_off(void);
unsigned short check_squelch(unsigned short port);
void rad_stop_scc(unsigned short port);
void rtc_tx_init(unsigned short port);
void rtc_rx_init(unsigned short port);
```

APPENDIX F

Program Listing Showing
Preferred Control Instructions
for the Baud Rate Switching Protocol
of Mobile Transceiver Unit 833

```
/* Copyright (c) 1991 Norand Corporation.  All rights reserved. */

/* Hardware independent MAC layer routines.  This module is responsible for
 * controlling and receiving messages from the physical hardware routines.
 * It fits in the protocol stack between the  LLC and the hardware specific
 * MAC layers.  This layer is responsible for supplying and removing the bloc
k
 * framing and check digits.
 */

/* $Log:   N:/VCS/PKTDRVR/MAC_RTC.C_V  $
 *
 *    Rev 1.57   07 Apr 1992 09:32:36   spiess
 * uses far model
 *
 *    Rev 1.56   21 Jan 1992 16:09:12   walterjo
 * Forgot to enable interrupts through all MACInit paths.
 *
 *    Rev 1.55   18 Jan 1992 11:02:46   spiess
 * Stop logging receive buffer overflow as an input error.
 *
 *    Rev 1.54   18 Jan 1992 09:52:42   spiess
 * Modified so that a shutdown is deferred until the transmitter has complete
d
 * its transmission.  If the shutdown is reversed before it has been executed
,
 * nothing wiIl happen if the transmitter has not stopped.
 *
 *    Rev 1.53   17 Jan 1992 13:35:36   spiess
 * Changed to not send up the carrier loss packet.  Defers shutdown if a
 * transmission is in progress.
 *
 *    Rev 1.52   27 Nov 1991 13:58:30   spiess
 * Doubled switch_delay for slow switches between abud rates.
 *
 *    Rev 1.51   14 Nov 1991 13:01:26   spiess
 * removed EAP conditional
 *
 *    Rev 1.50   01 Oct 1991 13:37:28   spiess
 * Lint.
 *
 *    Rev 1.49   24 Sep 1991 09:17:12   spiess
 * No change.
 *
 *    Rev 1.48   17 Sep 1991 08:49:52   spiess
 * EAP now appears functional
 *
 *    Rev 1.47   12 Sep 1991 12:02:12   spiess
 * Don't touch PTT when MACQuicker called.
 *
 *    Rev 1.46   10 Sep 1991 13:13:02   spiess
 * fixed bug in recovery from EAP to RTC
 *
 *    Rev 1.45   10 Sep 1991 11:00:52   spiess
 * lintability
 *
 *    Rev 1.44   06 Sep 1991 15:48:04   spiess
 * EAP modifications; Buffer management.
 *
 *    Rev 1.43   29 Aug 1991 14:00:02   spiess
 * Some progress toward EAP.
```

```
*
*    Rev 1.41   08 Aug 1991 13:09:34   spiess
* Changed goodpkt speed notification sequence.
*
*    Rev 1.40   08 Aug 1991 13:01:50   spiess
* Explicitly set State to START if restarting.
*
*    Rev 1.39   08 Aug 1991 12:15:00   spiess
* Fixed typo.
*
*    Rev 1.38   07 Aug 1991 11:59:18   spiess
* Notify app of carrier recovery.
*
*    Rev 1.37   03 Aug 1991 08:30:42   spiess
*
*    Rev 1.36   24 Jul 1991 13:48:16   spiess
* Prevent buffer overflow
*
*    Rev 1.35   22 Jul 1991 15:33:10   spiess
* Changed error value's return size.
*
*    Rev 1.34   11 Jul 1991 11:54:20   spiess
* Ignore uart errs altogether
*
*    Rev 1.33   11 Jul 1991 10:08:14   spiess
* The empirical best on ptt delay & Quicker
*
*    Rev 1.32   10 Jul 1991 12:46:48   spiess
* implement some _fastcall stuff, shorten ptt_delay2
*
*    Rev 1.31   10 Jul 1991 10:09:34   spiess
* Ptt delay & preamble.
*
*    Rev 1.30   09 Jul 1991 17:24:50   spiess
* PTT Delay
*
*    Rev 1.29   09 Jul 1991 17:05:52   spiess
* Removed preamble!!!
*
*    Rev 1.28   09 Jul 1991 15:50:22   spiess
* Ptt delay & preamble
*
*    Rev 1.27   09 Jul 1991 15:04:42   spiess
* Guess what?  Ptt delay & sync pattern
*
*    Rev 1.25   09 Jul 1991 13:24:20   spiess
* Used a new XmitNow() function
*
*    Rev 1.24   09 Jul 1991 12:32:56   spiess
* Changed preamble again
*
*    Rev 1.23   09 Jul 1991 11:59:30   spiess
* Changed preamble & PTT delay
*
*    Rev 1.22   09 Jul 1991 08:47:24   spiess
* Yet one more PTT delay change
*
*    Rev 1.21   08 Jul 1991 15:36:38   spiess
* Modified for dual xmit buffer.
*
*    Rev 1.20   08 Jul 1991 10:20:24   spiess
```

```
 * Changed ptt delay AGAIN!
 *
 *    Rev 1.19   08 Jul 1991 09:34:24   spiess
 * Another change to ptt delay
 *
 *    Rev 1.18   05 Jul 1991 16:45:44   spiess
 * Lengthen PTT (again)
 *
 *    Rev 1.17   05 Jul 1991 13:18:42   spiess
 * Attempted to modify preamble again
 *
 *    Rev 1.16   05 Jul 1991 13:03:56   spiess
 * Attempted to modify preamble again
 *
 *    Rev 1.15   05 Jul 1991 11:40:04   spiess
 * Restore ptt delay, change sync pattern.
 *
 *    Rev 1.14   04 Jul 1991 09:28:24   spiess
 *
 *    Rev 1.13   30 Jun 1991 08:57:58   spiess
 * Modified not to attempt to change the power on/off setting of the radio.
 * This is controlled by installing and uninstalling the uart layer.
 *
 *    Rev 1.12   20 Jun 1991 16:46:02   spiess
 * No change.
 *
 *    Rev 1.11   04 Jun 1991 10:02:58   spiess
 * Modified routines to start and stop mac layer to install and uninstall
 * the uart layer beneath, leaving the stop transmit and receive to that
 * layer.
 *
 *    Rev 1.10   22 May 1991 08:56:06   spiess
 * Changes after code review.
 *
 *    Rev 1.9   17 May 1991 16:08:06   walterjo
 * Now ignores uart framing errors.
 *
 *    Rev 1.8   17 May 1991 14:55:04   spiess
 * Fixed buffer overflow handling and output a 0x55 when starting a packet.
 *
 *    Rev 1.7   17 May 1991 08:15:02   spiess
 * Mostly added comments.  Made minor changes to disable the receiver while
 * the transmitter was running.
 *
 *    Rev 1.6   16 May 1991 10:19:34   spiess
 * A very fine version this is.  Seems to work well.
 *
 *    Rev 1.5   14 May 1991 13:36:46   spiess
 *
 *    Rev 1.4   14 May 1991 07:30:16   spiess
 * Version contains autobaud and more support routines.
 *
 *    Rev 1.3   09 May 1991 15:49:42   spiess
 * Added stuff to transmit RTC messages.  Added routine to get RTC bytes one
 * at a time.  Passed LLC layer the byte count without DLEs.
 *
 *    Rev 1.2   05 May 1991 16:22:14   spiess
 * No change.
 *
 *    Rev 1.1   02 May 1991 14:01:04   spiess
 * Many changes.  Discovered that we definitely need to split this
```

```
 * a MAC and LLC layer.  Several functions have been removed to llc_rtc.
 *
 *    Rev 1.0   29 Apr 1991 06:06:10   spiess
 * Initial revision.
 */

#define EAP_HEADER_LENGTH 3

#include <stdlib.h>
#include <dos.h>
#include <bios.h>
#include <string.h>
#include <nor/style.h>
#include <nor/mtask.h>
#include <nor/crc16.h>
#include "pktmac.h"
#include "pktdrvr.h"
#include "mac_rtc.h"
#include "llc_rtc.h"    /* so we can return SPEED_ to app */

#undef BUF_SIZE
#define BUF_SIZE (128*2+6+32)   /* stx+addr+command+128 transparent+etx+crc16
*/

/*****************************************************************************
**
 * Autobaud switch
 */
#define SWITCH_UP_DELAY         20      /* # Pkts good before rate speedup */
#define CARRIER_TRIES           6       /* switch rates this many times */
#define SWITCH_SEEK_DELAY       1024    /* # bytes bad before rate switch */
#define SWITCH_DOWN_DELAY       CARRIER_TRIES*SWITCH_SEEK_DELAY
                                /* # bytes bad before rate slqwdown *
/
/* Autobaud variables */
    EXPORT int AutoBaud = 1;            /* 0=no, 1=yes */
    EXPORT int HighSpeed = 0;           /* 0=4800, 1=9600 */
    EXPORT int _far InvertPTT = 0;              /* 0=Normal, 1=direct to cont
roller */
    PRIVATE unsigned BadDelay = SWITCH_SEEK_DELAY;/* bad chars to go */
    PRIVATE unsigned GoodDelay = SWITCH_UP_DELAY;/* good packets to go */
    PRIVATE int CarrierLoss = CARRIER_TRIES;/* did we already notify llc? */
    PRIVATE unsigned Quicker = 0;               /* ptt early warning variable
 */

/*****************************************************************************
**
 * Hardware specific routines to control a serial line
 *
 * The push to talk delay should be no less than 6 milliseconds.  After that
 * time, the receiver needs conditioning so that its detection circuitry can
 * tell the difference between a zero and a one.  There must also be a quiet
 * time of some sort for the remote uart to synchronize with the start bit.
 * all together a delay of 14 milliseconds before the first data character
 * is appropriate.
 */
    IMPORT void UartSetup(unsigned rate, int Control);
    IMPORT void UartSignals(unsigned char ChangeThese,unsigned char ThisValue
);
#define PUSH_TO_TALK 2                  /* same as RTS! */
#define PTT_DELAY1 7                    /* Milliseconds before ready to send
```

```
 */
#define PTT_DELAY2 7                    /* Milliseconds before ready to send
 */
#define FAST_SLOT_DELAY 25              /* Milliseconds per slot @ 9600 */
#define SLOW_SLOT_DELAY 40              /* Milliseconds per slot @ 4800 */

#define PTT_DELAY_EAP 45                /* Milliseconds before ready to send
 */

/*****************************************************************************
**
 * Block framing characters
 */

#define STX             2               /* Start of TeXt */
#define ETX             3               /* End of TeXt */
#define DLE             0x10            /* Data Link Escape */

/*****************************************************************************
**
 * Receiver variables
 */
    PRIVATE unsigned   AlreadyHere = 0; /* Mutual exclusion semaphore */
    PRIVATE enum Protocols RecvProtocol = USE_RTC;
    PRIVATE enum {                      /* frame transfer States. */
        START = 0,                      /* STX */
        END   = 1,                      /* ETX */
        CRC1  = 2,                      /* first CRC character */
        CRC2  = 3,                      /* last CRC character */
        DLESTX= 4,
        HEADER= 5,
        DATA  = 6,
        LRC   = 7,
        HASH  = 8
    } State = START;                    /* what comes next? */
    unsigned char RBuf[BUF_SIZE];
    static unsigned RBufLength = 0;
    unsigned char *RBufPtr = RBuf;

/*****************************************************************************
**
 * Transmitter variables
 */
    PRIVATE int TransmitterRunning = 0; /* state of transmitter */
    PRIVATE int PendingShutdown = 0;    /* state of trying to reset */
    PRIVATE int ParityEven(int c);

PRIVATE void CommitRecv(void) {
/*****************************************************************************
**
 * Causes the receive data that has already been obtained to be removed from
 * the buffer.
 */
    if (RBuf+RBufLength > RBufPtr) {
        RBufLength -= (unsigned)(RBufPtr-RBuf);
        memcpy(RBuf, RBufPtr, RBufLength);
    } else {
        RBufLength = 0;
    }
    RBufPtr = RBuf;
}
```

```
PRIVATE void _fastcall GoodPkt(void) {
/*****************************************************************************
**
 * Increases the baud rate after good experiance at the current rate.
 * resets the carrier loss indication.
 */
    if (!CarrierLoss) {                         /* notify app of carrier */
        (void)LogicalLinkControl(
            0,
            (unsigned char)(HighSpeed?
                (RecvProtocol == USE_EAP)? SPEED_9600_EAP :
                SPEED_9600_RTC
            :
                (RecvProtocol == USE_EAP)? SPEED_4800_EAP :
                SPEED_4800_RTC
            )
        );
    }
    CarrierLoss = CARRIER_TRIES;                /* we have carrier */
    BadDelay = SWITCH_DOWN_DELAY;               /* take a while before slow *
/
    if (AutoBaud && !HighSpeed) {
        if (GoodDelay) {                        /* if low speed */
            --GoodDelay;                        /* advance toward high speed
*/
        } else if (!TransmitterRunning) {
            HighSpeed = 1;
            UartSetup(96, _COM_CHR8 | _COM_STOP1 | _COM_NOPARITY);
            CarrierLoss = 0;                    /* force app notification */
            BadDelay = SWITCH_SEEK_DELAY;       /* don't try too hard */
        }
    }
}

PRIVATE void _fastcall BadBytes(unsigned Quantity) {
/*****************************************************************************
**
 * Decreases the baud rate after bad experiance at the current rate.
 * if we fail at 4800 baud, or are not auto switching then notify app of
 * comm loss.
 */
    if (BadDelay > Quantity) {/* if not bad enough experiance */
        BadDelay -= Quantity;                   /* reduce toward low speed */
    } else {
        GoodDelay = SWITCH_UP_DELAY;
        BadDelay = SWITCH_SEEK_DELAY;
        if (AutoBaud) {
            HighSpeed = !HighSpeed;
            UartSetup(
                HighSpeed ? 96 : 48,
                _COM_CHR8 | _COM_STOP1 | _COM_NOPARITY
            );
        }
        if (CarrierLoss && !--CarrierLoss) {    /* notify app of carrier loss
*/
            (void)LogicalLinkControl(0, SPEED_LOSS);
        }
    }
}
```

```
EXPORT void PutRecvError(unsigned char c, unsigned char Error) {
/*****************************************************************************
**
 * We just received an error.  Soft flush the receive buffer by inserting a
 * STX if a hardware error occurred.  If the buffer overflowed and parsing
 * is not currently in progress, throw away the oldest character and add the
 * new character.  If the buffer has overflowed and parsing is in progress,
 * we assume that a message may be pulled out of the buffer soon and discard
 * the new character.
 */
    Error = Error;
    if (!PutRecvByte(c)) {
        if (!AlreadyHere) {              /* if not already processing */
            State = START;
            ++RBufPtr;                   /* get at least one more character */
            CommitRecv();                /* clear some stuff out */
            (void)PutRecvByte(c);        /* put the new character in */
        }
    }
}

PRIVATE void AcceptBlock(
    unsigned char Type,
    unsigned PktLength,
    unsigned JnkLength
) {
/*****************************************************************************
**
 */
    Quicker = 0;
    if (LogicalLinkControl(PktLength, Type)) {  /* fwd pkt & speed */
        GoodPkt();                       /* good pkt from host*/
    }
    RBufPtr = RBuf+PktLength+JnkLength;
}

EXPORT void InterruptComplete(int EndTransmit) {
/*****************************************************************************
**
 * This routine is called after a uart interrupt.  It attempts to assemble an
d
 * validate a received packet so that it can be forwarded to the Logical Link
 * Layer.
 */
    PRIVATE unsigned PktLength = 0;      /* Length of text */
    PRIVATE unsigned JnkLength = 0;      /* Length of text */
    PRIVATE unsigned Crc = 0;            /* accumulation bucket */
    int c;                               /* character received */
    PRIVATE unsigned char Hash = 0;              /* accumulation bucket */
    PRIVATE unsigned char Lrc = 0;               /* accumulation bucket */

    /* update state of transmitter */
    if (TransmitterRunning) {
        if (EndTransmit) {
            UartSignals(PUSH_TO_TALK, (unsigned char)(InvertPTT ? ~0 : 0));
            TransmitterRunning = 0;
            if (PendingShutdown) {
                MACReset();
            } else {
                StartReceiver();
            }
```

```
        }
        return;
    }

    /* Provide mutual exclusion on reentry so we can allow interrupts */
    if (AlreadyHere) return;

    /* Process all characters received so far. New characters may be added
     * while we're in this routine!
     */
    for (;;) {

        /* Get a character and set AlreadyHere in one indivisible operation
         * (meaning interrupts are disabled).  This lets anyone reentering
         * know if we're finished or not, preventing interleaved receive
         * character processing.  We won't enable interrupts after clearing
         * the mutual exclusion variable to prevent stack runaway.
         */
        _disable();
        AlreadyHere = (RBufPtr < RBuf+RBufLength);
        _enable();
        if (!AlreadyHere) break;
        c = *RBufPtr++;

        switch (State) {
            default     :                   /* just in case */
                State = START;                          /* reset state */
            case(START) :                   /* looking for start of text */
                if (c == DLE) {
                    State = DLESTX;
                } else
                if (c == STX) {
                    State = END;                        /* advance state */
                    PktLength = 0;                      /* no length yet */
                    JnkLength = 0;                      /* #Overhead bytes */
                    Crc = 0;                    /* prime CRC */
                } else {
                    BadBytes(1);
                }
                CommitRecv();                   /* dispose trash */
                break;

            case(END)   :                   /* looking for end of text */
                if (c == STX) {
                    State = START;
                    --RBufPtr;
                    if (JnkLength && RBufPtr[-1] == DLE) {
                        State = DLESTX;
                        JnkLength--;
                    }
                    BadBytes(PktLength + JnkLength);
                } else {
                    Crc = CrcByte(c, Crc);              /* accum CRC */
                    if (c == ETX) {
                        State = CRC1;                   /* advance state */
                        JnkLength++;                    /* increase length */
                    } else if (c == DLE) {
                        JnkLength++;                    /* increase length */
                    } else {
                        PktLength++;                    /* increase length */
                    }
```

```
        }
        break;

    case(CRC1)  :                   /* this is first CRC character */
        Crc = CrcByte(c, Crc);                      /* accum CRC */
        State = CRC2;                               /* advance state */
        JnkLength++;                                /* increase length */
        break;

    case(CRC2)  :                   /* this is second CRC character */
        Crc = CrcByte(c, Crc);                      /* accum CRC */
        State = START;                              /* restart state */
        JnkLength++;                                /* increase length */
        if (!Crc) {
            RecvProtocol = USE_RTC;
            AcceptBlock(
                (unsigned char)(
                    HighSpeed? SPEED_9600_RTC : SPEED_4800_RTC
                ),
                PktLength,
                JnkLength
            );
        } else {
            BadBytes(PktLength + JnkLength);        /* tally bad */
        }
        break;

    case(DLESTX):
        if (c == STX) {
            State = HEADER;
            Lrc = 0;
            Hash = 0;
            JnkLength = EAP_HEADER_LENGTH;
        } else {
            State = START;
            BadBytes(2);
        }
        CommitRecv();
        break;

    case(HEADER):
        if (ParityEven(c)) {
            State = END;
            Crc = 0;                                /* prime CRC */
            PktLength = 0;                          /* no length yet */
            JnkLength = 0;                          /* #Overhead bytes */
            RBufPtr = RBuf;
        } else {
            Lrc ^= (unsigned char)c;
            Hash += (unsigned char)c;
            if (!--JnkLength) {
                PktLength = c & 0x7f;
                PktLength++;
                State = DATA;
            }
        }
        break;

    case(DATA)  :
        if (ParityEven(c)) {
            State = END;
```

```
                    Crc = 0;                                    /* prime CRC */
                    PktLength = 0;                              /* no length yet */
                    JnkLength = 0;                              /* #Overhead bytes */
                    RBufPtr = RBuf;
                } else {
                    Lrc ^= (unsigned char)c;
                    Hash += (unsigned char)c;
                    if (++JnkLength >= PktLength) {
                        State = LRC;
                        PktLength += EAP_HEADER_LENGTH;
                    }
                }
                break;

            case(LRC)    :
                if (ParityEven(c)) {
                    State = END;
                    Crc = 0;                                    /* prime CRC */
                    PktLength = 0;                              /* no length yet */
                    JnkLength = 0;                              /* #Overhead bytes */
                    RBufPtr = RBuf;
                } else {
                    Lrc ^= (unsigned char)c;
                    State = HASH;
                }
                break;

            case(HASH)   :
                if (
                    ParityEven(c) ||
                    (Lrc & 0x7f) ||
                    (Hash & 0x7f) != (c & 0x7f)
                ) {
                    State = END;
                    Crc = 0;                                    /* prime CRC */
                    PktLength = 0;                              /* no length yet */
                    JnkLength = 0;                              /* #Overhead bytes */
                    RBufPtr = RBuf;
                } else {
                    State = START;
                    RecvProtocol = USE_EAP;
                    AcceptBlock(
                        (unsigned char)(
                            HighSpeed? SPEED_9600_EAP : SPEED_4800_EAP
                        ),
                        PktLength,
                        2
                    );
                }
                break;
        }
    }
}

EXPORT int PutRecvByte(unsigned char c) {
/*****************************************************************************
**
 * Place received byte into the appropriate receive buffer.
 * If the head pointer catches the tail pointer, we lose this character and
 * return zero to indicate failure.
 */
```

```
    if (RBufLength >= sizeof(RBuf)) {
        return 0;
    } else {
        RBuf[RBufLength++] = c;
        return 1;
    }
}

EXPORT unsigned char MACInByte(void) {
/*****************************************************************************
**
 * Handle the receive character which may possibly be modified with a DLE.
 * We could return an End Of Buffer indication to the caller when we receive
 * one from GetRecvByte, but the LLC knows the message length and doesn't
 * expect errors from here.
 */
    unsigned char c;

    if (RecvProtocol == USE_RTC) {
        c = *RBufPtr++;
        if (c != DLE) {
            return c;
        }
    }
    return (unsigned char)(*RBufPtr++ & 0x7f);
}

EXPORT void MACKeyUp(int FastTerminal) {
/*****************************************************************************
**
 * start transmitting data
 */
    TransmitterRunning = 1;
    StopReceiver();
    UartSignals(PUSH_TO_TALK, (unsigned char)(InvertPTT? 0:~0));
    if (!FastTerminal) {
        UartDelay(PTT_DELAY_EAP);
    } else {
        UartDelay(PTT_DELAY1 - Quicker);        /* push to talk delay */
        XmitCharNow(0xf);
        UartDelay(PTT_DELAY2);                  /* push to talk delay */
    }
    StartTransmitter();                         /* squeak only one character out */
}

EXPORT void MACDelay(unsigned Slots) {
/*****************************************************************************
**
 * delay # slots.
 */
    UartDelay(Slots * (HighSpeed ? FAST_SLOT_DELAY : SLOW_SLOT_DELAY));
}

EXPORT void MACReset(void) {
/*****************************************************************************
**
 * Turn it all off.
 */
    PendingShutdown = 1;
    if (!TransmitterRunning) {
        PendingShutdown = 0;
```

```
        UninstallCom();
    }
}

EXPORT void MACQuicker(void) {
/*****************************************************************************
**
 * Faster ptt processing will be required for the next transmission.
 */
    Quicker = 2;                            /* assume 2 ms passed in MACKeyUp */
    /* StopReceiver(); */
    /* UartSignals(PUSH_TO_TALK, (unsigned char)(InvertPTT? 0:~0)); */
}

EXPORT char MACInit(unsigned Port) {
/*****************************************************************************
**
 * turn it all on
 */
    _disable();
    if (PendingShutdown || TransmitterRunning) {
        PendingShutdown = 0;
        _enable();
    } else {
        _enable();
        MACReset();
        if (InstallCom(Port) < 0) return NO_NUMBER;
        UartSetup(HighSpeed ? 96 : 48, _COM_CHR8 | _COM_STOP1 | _COM_NOPARITY
);
        UartSignals(PUSH_TO_TALK, (unsigned char)(InvertPTT? ~0:0));
        StartReceiver();
    }
    return 0;
}

#pragma optimize("leg", off)

PRIVATE int ParityEven(int c) {
    _asm {
        xor     ax,ax
        or      ax,[c]
        lahf
        and     ax,400h
        mov     [c],ax
    }
    return c;
}
```

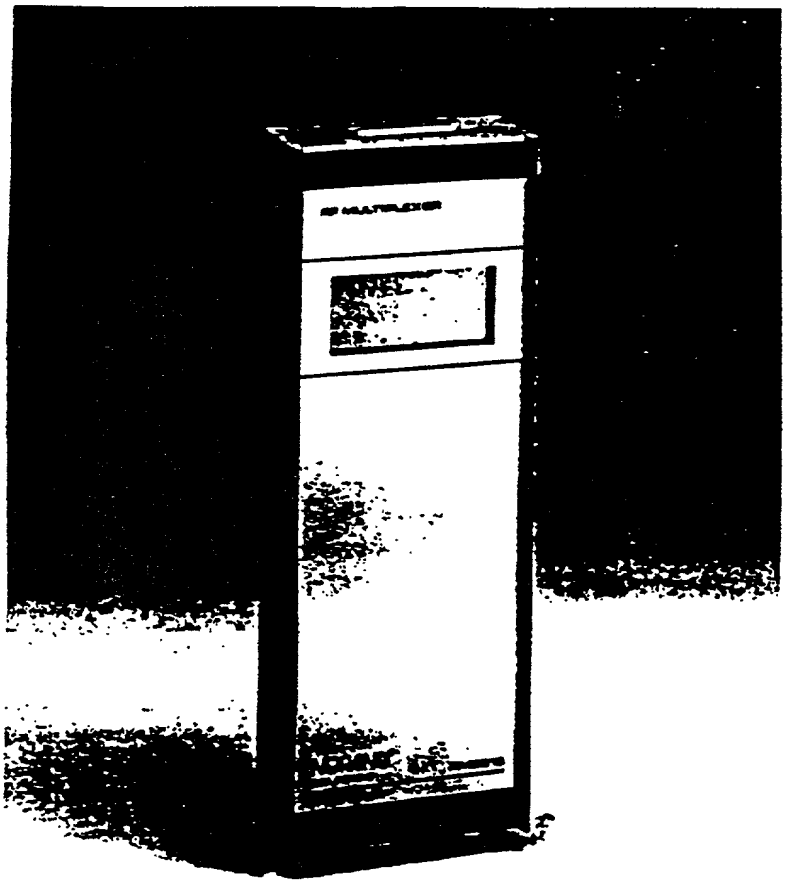
APPENDIX G

One-page brochure of Norand Corporation, Cedar Rapids, Iowa, entitled "RM3216 Communication Multiplexer", (Copyright Norand Corporation 1990 All Rights Reserved)

NORAND®
DATA SYSTEMS

RM3216
Communications Multiplexer

Introducing the New NORAND® RM3216 Communications Multiplexer



Features:

- Enhanced Adaptive Polled Protocol™ for rapid response time
- 620 byte data packetization:
  Large screen support
  Tabling and printing support
- Compatible with all current production models of Norand® hand-held radio data terminals
- Supports up to 16 radio data terminals simultaneously
- RS-232 and RS-422 host interface capability
- Host data communications
  2,400; 4,800; 9,600 or 19,200 b.p.s.
- Liquid crystal display (4 line x 16 character) for diagnostic information and status of radio terminals
- Power source
  NORAND NC3000 Power Supply
- Program memory
  Flash ROM
- Static shock protection
  Hardened against electrostatic discharges up to 16.5 KV
- Physical Characteristics
  Length: 9.3 inches
  Width: 3.3 inches
  Depth: 1.6 inches
  Weight: 1.3 pounds
- Operating temperature:
  32° to 104°F
  (0° to 40°C)
- Storage temperature:
  -40° to 158°F.
  (-40° to 70°C)

Norand Corporation
550 Second Street S.E.
Cedar Rapids, Iowa 52401
Phone: 319-369-3156
1-800-553-5971 toll free (ext. 3156)

The goal of Norand is
100% customer satisfaction.
Customer Satisfaction Hot Line:
1-800-221-9236

960-330-010 Printed in U.S.A.

APPENDIX H

Two-page listing of command structure for dormant/active cycling of mobile transceiver unit, (Copyright Norand Corporation 1991 All Rights Reserved)

```
;
;  Equates for the timer reload values
;
RAD_ON_TM1      EQU     10              ;RADIO POWER "ON" 10 SECOND CYCLE TIME
RAD_ON_TM2      EQU     1               ;RADIO POWER "ON" SECOND CYCLE TIME
RAD_OFF_TM1     EQU     5               ;RADIO POWER "OFF" 5 SECOND TIME

;-----------------------------------------------------------------------

; This routine is called once per second to handle the radio power
; cycling.  If in link test mode (for NORAND test purposes only)
; then the whole algorithm is disabled and the radio is always on.
;
; Function:
;
;   START:
;       Set radio on for 10 seconds
;       If no communication with host or scanning then
;         Radio off for 5 seconds
;         Radio on for 1 second
;         Radio off for 5 seconds
;         Radio on for 1 second
;           .
;           .
;           .
;
;       Else if communication with host or scanning occurs
;         Reset (restart) radio power on cycle for 10 seconds (Jump to START)
;
;
;
RADIO_POWER:
        JB      RECHRG,RADP_020         ; IF NOT CHARGING THEN
        JNB     LNKTST,RADP_000         ;  IF LINK TEST MODE ENABLED THEN
        LCALL   RESTORE_RADIO           ;   TURN ON THE RADIO
        SJMP    RADP_020                ;   AND EXIT
RADP_000:
        MOV     DPTR,#RADIO_TMR1        ;   GET RADIO ON POWER TIMER
        MOVX    A,@DPTR                 ;
        JZ      RADP_005                ;   IF NOT ZERO THEN
        DEC     A                       ;     DECREMENT THE TIMER
        MOVX    @DPTR,A                 ;
        LCALL   RESTORE_RADIO           ;     TURN ON THE RADIO
        SJMP    RADP_020                ;
RADP_005:
        SETB    RADIO_PWR               ;   TURN OFF RADIO POWER
        CLR     TXD                     ;   MAKE TXD LINE LOW WHILE RADIO OFF
        MOV     DPTR,#RADIO_TMR2        ;   GET RADIO OFF POWER TIMER
        MOVX    A,@DPTR                 ;
        JNZ     RADP_010                ;   IF TIMER = 0
        LCALL   RESTORE_RADIO           ;     TURN ON RADIO POWER
        MOV     A,#RAD_OFF_TM1          ;     RE-LOAD RADIO OFF POWER TIMER
        MOVX    @DPTR,A                 ;     (5 SECONDS)
        MOV     DPTR,#RADIO_TMR1        ;     RE-LOAD RADIO ON POWER TIMER
        MOV     A,#RAD_ON_TM2           ;     (1 SECOND)
        MOVX    @DPTR,A                 ;
        SJMP    RADP_020                ;
RADP_010:                               ;   ELSE RADIO OFF TIMER <> 0
        DEC     A                       ;     DEC POWER TIMER
        MOVX    @DPTR,A                 ;     TURN RADIO OFF
        SETB    RADIO_PWR               ;
        CLR     TXD                     ;     MAKE TXD LINE LOW WHILE RADIO OFF
RADP_020:
        RET

;-----------------------------------------------------------------------
;
;  This routine is called to reset the radio "on" power cycle.
;  It is called whenever a barcode is scanned, data is sent to the
;  host from the terminal, or when the terminal receives a message
;  from the host.
;
;ROUTINE NAME: RADIO_ON
```

```
;
;PURPOSE:
;         THIS ROUTINE TURNS ON THE RADIO POWER AND ALSO RELOADS THE
;         RADIO POWER CYCLE TIMER.
;
;INPUTS:  NONE
;
;OUTPUTS: NONE
;
;REGISTERS USED: ACCUMULATOR, DPTR
;
;CALLED BY: FMSEND, RDRADIO, RESTART
;
;CALLS: NONE
;
RADIO_ON:
          CLR     RADIO_PWR           ;TURN ON RADIO
          PUSH    ACC                 ;
          PUSH    DPH                 ;
          PUSH    DPL                 ;
          MOV     DPTR,#RADIO_TMR1    ;LOAD RADIO ON POWER TIMER
          MOV     A,#RAD_ON_TM1       ;
          MOVX    @DPTR,A             ;
          MOV     DPTR,#RADIO_TMR2    ;LOAD RADIO OFF POWER TIMER
          MOV     A,#RAD_OFF_TM1      ;
          MOVX    @DPTR,A             ;RESET RADIO POWER TIMER
          POP     DPL                 ;
          POP     DPH                 ;
          POP     ACC                 ;
          RET

;-------------------------------------------------------------------------
;
RESTORE_RADIO:
        SETB    TXD                   ; RETURN TXD LINE TO HIGH STATE
        MOV     C,SPEED_BIT           ; MOVE STATE OF SPEED_BIT IN C
        CPL     C                     ;
        MOV     BAUDFRQ,C             ; BAUDFRQ = /SPEED_BIT
        CLR     RADIO_PWR             ; TURN ON THE RADIO
        RET
;
;-------------------------------------------------------------------------
```